Markt&Technik

Holger Gzella

AMIGA MODULA-2

Programmieren für Fortgeschrittene

DOS-Programmierung ★ Intuition ★ Grafik ★ Copper ★ Blitter ★ Diskfonts ★ Exec ★ Tips und Tricks



**Auf 3½"-Diskette enthalten:
Alle im Buch aufgeführten Beispiele als Source-Code und ablauffähiger Version.**

AMIGA Modula-2

Modula-2 ist eine Hochsprache, die systemnahe Programmierung unterstützt. Dieses Lehr- und Nachschlagebuch ist für diejenigen geschrieben, die Modula-2 bereits in seinen Grundzügen kennen und nun wissen möchten, wie man die speziellen Eigenschaften des Amiga ausreizt.
Erfahrungsgemäß bereitet die Programmierung des Amiga besondere Schwierigkeiten: Das System ist zwar extrem leistungsfähig, es stellt aber auch höchste Ansprüche an die Programmierer. Der Autor beschäftigt sich daher zunächst ausführlich mit DOS und »Intuition«.
Der Name Amiga ist eng mit der Erzeugung faszinierender Grafik verbunden; deshalb macht die Grafikprogrammierung einen zweiten Großteil des Buches aus: Sie erfahren dort alles über »Graphics«, »Copper« und »Blitter«.

Im dritten Teil des Buches beschäftigt sich der Autor mit »Exec«: das Multitasking und die Devices des Amiga sowie deren Programmierung unter Modula-2 stehen dabei im Vordergrund.

Im Anhang finden Sie lehrreiche Tips und Tricks, die Ihnen die Arbeit mit den Compilern erleichtern. Aus dem Inhalt:

- Programmierung von Intuition, Ausstatten eigener Programme mit Pull-down-Menüs, Gadgets, Requester
- Grafikprogrammierung mit den Amiga-Befehlen
- Hardware-Sprites, virtuelle Sprites und Bobs
- Grafik-Spezialmodi HAM (4096 Farben gleichzeitig) und Halfbrite (64 Farben gleichzeitig)
- Dual-Playfields und Soft-Scrolling
- Layers und Superlayers
- IFF-Standard
- Copper- und Blitterprogrammierung
- Amiga-Zeichensätze
- Programmierung des Multitasking-Systems
- Programmierung der wichtigsten Devices
- Programmierung des Diskettenbetriebssystems
- Tips und Tricks

Hardware-Anforderungen:

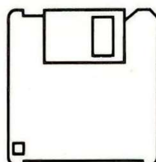
- Amiga 500, 1000 oder 2000 mit mindestens 512 Kbyte RAM (ab Kickstart Version 1.2), Drucker empfohlen

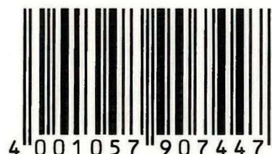
Software-Anforderungen:

- Modula-2-Compiler, etwa den M2Amiga

ISB N 3-89090-744-X







DM 69,-
sFr 63,50
öS 538,-

Holger Gzella

AMIGA Modula-2

Programmieren für Fortgeschrittene

DOS-Programmierung
* Intuition * Grafik * Copper
* Blitter * Diskfonts * Exec
* Tips & Tricks

Markt & Technik
Verlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Gzella, Holger:
AMIGA Modula-2: Programmieren für Fortgeschrittene;
DOS-Programmierung, Intuition, Grafik, Cooper, Blitter,
Diskfonts, Exec, Tips und Tricks / Holger Gzella.
Haar bei München: Markt&Technik-Verl., 1989
ISBN 3-89090-744-X

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

92 91 90 89

ISBN 3-89090-744-X

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Druckerei Mühlberger GmbH, Gersthofen
Printed in Germany

Inhaltsverzeichnis

Vorwort

Vor kurzem erst hat die verhältnismäßig junge Computersprache MODULA-2 auf dem Amiga Einzug gehalten, um C Paroli zu bieten. Doch entsetzt muß der künftige MODULA-2-Programmierer feststellen, daß das Angebot an Literatur, die für vernünftiges Programmieren unentbehrlich ist, nicht gerade wohl bestückt ist. Sicher gibt es einige Werke, mit denen man MODULA-2 lernen kann, doch möchte man den Amiga ausnutzen und systemnah programmieren, ist man, zumindest zum Zeitpunkt der Erstellung dieses Buches, ziemlich alleingelassen. Die meisten Programmierbücher für den Amiga setzen Kenntnis in der Sprache C, manchmal auch in der englischen Sprache voraus.

So kam es, daß ich mich eines Sommertages, als ich mich bemühte, meine ersten Programme mit dem neuen MODULA-2-Compiler zu schreiben, furchtbar ärgerte, daß es keine unterstützende Literatur gab, ist MODULA-2 doch eine wunderbar strukturierte Hochsprache, mit der sich etliche Anwendungen realisieren lassen. Einige Zeit später faßte ich den Entschluß, etwas gegen diesen unmöglichen Zustand zu tun. So schrieb ich dieses Buch, um Ihnen die nötigen Informationen für eine konsequente Ausnutzung des Amiga zu geben und versuchte, ein möglichst breites Themenspektrum in Buchform zu packen.

Herausgekommen ist das, was Sie in Händen halten – ein Lehrbuch wie Nachschlagewerk für den fortgeschrittenen Programmierer, der MODULA-2 beherrscht und sich nun an die Systemprogrammierung des Amiga machen will.

Ich hoffe, dieses Buch vermittelt Ihnen einen leichten Zutritt zu den fantastischen Fähigkeiten des Amiga und wird in der nächsten Zeit sicher in unmittelbarer Nähe Ihres Computers liegen, aber Ihnen auch später in manchen Fragen helfen.

Bis dahin, viel Spaß beim Lesen!

Holger Gzella

Kapitel 1

Einführung

1.1 Einführung in die Systemarchitektur des Amiga

Die Betriebssystemsoftware des Amiga besteht aus zahlreichen, kleineren Routinen, die sich, thematisch geordnet, in diversen Bibliotheken (Libraries) befinden. Die relevanten Libraries befinden sich im ROM (Amiga 500/2000) oder auf der Kickstart-Diskette (Amiga 1000), so auch die EXEC-Bibliothek, die die elementaren Routinen des Betriebssystems enthält. Die weniger wichtigen (zum Beispiel DISKFONT, mit der auf die verschiedenen Zeichensätze im Verzeichnis Fonts der Startdiskette zugegriffen werden kann) befinden sich (hoffentlich) auf der Workbench- oder Startdiskette. Mit diesem Konzept wird dem Programmierer sehr viel Arbeit abgenommen.

1.2 Zugriff auf die Bibliotheken

Um Zugriff zu einer Library haben zu können, muß diese erst geöffnet und am Programmende wieder geschlossen werden, wenn man von EXEC, die von der Programmstartup-Routine geöffnet wird, absieht. In ihr sind auch die beiden Funktionen OPENLIBRARY und CLOSELIBRARY enthalten, die benötigt werden, um andere Bibliotheken zu öffnen. Das Laufzeitsystem des MODULA-2-Compilers M2Amiga von Meier-Vogt, mit dem ich arbeite und auf den ich mich auch im Verlauf des Buches beziehe, öffnet aber schon alle benötigten Libraries und schließt sie am Programmende auch wieder, so daß sich der Programmierer darum nicht kümmern muß, anders als z.B. beim TDI- oder Benchmark-Modula-2. Aber um die Bibliotheksroutinen aufrufen zu können, braucht jede Hochsprache spezielle Listen, in denen die Syntax der Funktion und gegebenenfalls auch wichtige und oft gebrauchte Konstanten definiert sind. In Basic werden dazu die »FD«-Files, in C die Libs und die Includes und in Modula-2 die System-Module benötigt. Diese Module (sie sind im M2Amiga-Handbuch ab Seite 103 abgedruckt) sind nichts anderes als Definitionsmodule, von denen wir die gesamten Typen, Konstanten und Prozeduren, die unser Programm braucht, importieren können (bzw. müssen). Besagte Definitionsmodule sind inhaltsmäßig teilweise mit den

Includes in C, die im ROM-KERNEL-MANUAL Band 1+2 abgedruckt sind, identisch, bis auf die Tatsache, daß seltsamerweise die Namen diverser Felder und Konstanten geändert und einige völlig weggelassen wurden. (Eine vermeidbare Fehlerquelle, der sich die Programmierer von M2Amiga hoffentlich bewußt sind.)

So würde ein Programm, das die Prozedur DELAY aus der DOS-Library, die einen angegebenen LONGINT-Wert in 1/50stel-Sekunden wartet, folgendermaßen aussehen:

```
MODULE DelayDemo;

FROM Dos IMPORT Delay;

BEGIN
  Delay(50) (* 50 IntuiTicks (1 IntuiTick = 1/50 Sekunde) warten *)
END DelayDemo.
```

Kapitel 2

Das DOS

Das DOS ist die Schnittstelle zwischen Benutzer und physikalischem Gerät (alles, was vom Tisch fallen kann). Die Aufgaben des DOS (Disk Operating System, also Diskettenbetriebssystem) sind Dinge wie die Kommunikation zwischen Benutzer und Hardware (Laufwerke, Drucker, ...). Das Ausdrucken eines Textes oder das Laden eines Programms erfolgt über das DOS. In diesem Kapitel soll besprochen werden, wie das DOS zu programmieren ist.

2.1 Organisation der DOS-Befehle

Die DOS-Befehle sind in der »dos.library«, die sich im ROM (oder beim Amiga 1000 auf der Kickstart-Disk) befindet, versammelt. Im Gegensatz zu den C-Programmierern oder Benutzern des TDI-Compilers brauchen die glücklichen Besitzer des M2Amiga-Compilers sich nicht um Öffnen und Schließen der Libraries zu kümmern. Sollten Sie mit einem TDI- oder Benchmark-Compiler arbeiten, müssen Sie das Öffnen der Library zu Anfang des Hauptprogramms mit dem Befehl OPENLIBRARY selbst vornehmen. Welche Parameter er benötigt, erfahren Sie aus dem EXEC-Kapitel.

2.2 Die ersten Befehle

Doch wir wollen uns nicht allzulange mit trockener Theorie aufhalten und uns sogleich den ersten Befehlen zuwenden. Als erster sei der WRITE-Befehl genannt, mit dem man Texte in Dateien oder auf den Bildschirm schreiben kann. Seine Syntax lautet folgendermaßen:

ok := Write (FileHandle , Text , Anzahl);

ok	LONGINT-Rückgabewert; ist er gleich 1, so wurde der WRITE-Befehl fehlerfrei ausgeführt, ist er aber 0, so trat irgendein Fehler auf.

FileHandle	Dateizeiger vom Typ »FileHandlePtr«, der auf die Datei zeigt, in die man mit WRITE schreiben will, doch dazu später.
Text	Der eigenliche Text, bzw. ein Zeiger darauf. Er muß vom Typ »ADDRESS« sein.
Anzahl	Anzahl der auszugebenden Zeichen, Typ LONGINT.

Will man Text im CLI-Fenster, dessen Dateizeiger man natürlich nicht kennt, ausgeben, so gibt es eine einfache Möglichkeit, sich diesen zu besorgen, nämlich den Befehl OUTPUT:

FileHandle := Output();

Jetzt zeigt »FileHandle« auf das aktuelle CLI-Fenster, d.h. das CLI-Fenster, aus dem das Programm, welches den OUTPUT-Befehl verwendet, gestartet wurde. OUTPUT liefert aber nur den Zeiger auf den aktuellen Ausgabekanal zurück. Möchte man aber den Eingabekanal wissen, so kann man sich dessen Handle mit folgendem Befehl besorgen:

FileHandle := Input();

Anhand eines kurzen Beispiels möchte ich den Read- und den Write-Befehl demonstrieren:

```
MODULE WriteDemo;

FROM SYSTEM  IMPORT ADR;
FROM Dos    IMPORT Write,Output,FileHandlePtr;

VAR MeinHandle: FileHandlePtr;
  ok     : LONGINT;

BEGIN
 MeinHandle:=Output();         (* Adresse des CLI-Fensters holen *)
 ok:=Write(MeinHandle,ADR("*** H A L L O ! ***"),19); (* schreiben *)
END WriteDemo.
```

Übrigens: einen Zeilenvorschub kann man durchführen, indem man das Zeichen »12C« ausgeben läßt:

```
 VAR lf : CHAR;
  ...

 lf:=12C
 ok:=Write(MeinHandle,ADR(lf),1);
```

Fügen Sie diese Zeilen in das obige Programm ein und staunen Sie!

Wenden wir uns dem READ-Befehl zu, mit dem man Zeichen aus einer Datei oder von der Tastatur lesen kann:

gelesen := Read (FileHandle , String , maxAnzahl);

FileHandle	Wie beim WRITE-Befehl der Dateizeiger vom Typ FileHandlePtr.
String	Zeiger auf einen String (vom Typ ADDRESS), in den die eingelesene Zeichenkette geschrieben wird.
maxAnzahl	Anzahl der maximal einzulesenden Zeichen.
gelesen	Anzahl der tatsächlich gelesenen Zeichen.

Ist »gelesen« gleich 0, so ist das Dateiende erreicht, tritt NIL(–1) auf, so ist irgend etwas schief gelaufen. Auch hierzu ein Beispielprogramm, und zwar eins von der klassischen Sorte: Sie geben Ihren Namen ein und das Programm begrüßt Sie entsprechend:

```
MODULE NameDemo;

FROM SYSTEM  IMPORT ADR;
FROM Dos     IMPORT Read,Write,Output,FileHandlePtr;

VAR MeinHandle: FileHandlePtr;
  ok,anzahl   : LONGINT;
  Name        : ARRAY[0..79] OF CHAR;è
BEGIN
 MeinHandle:=Output();            (* Adresse des CLI-Fensters holen *)
 ok:=Write(MeinHandle,ADR("Geben Sie Ihren Namen ein: "),27); (* schreiben *)
 anzahl:=Read(MeinHandle,ADR(Name),SIZE(Name));              (* einlesen *)
 ok:=Write(MeinHandle,ADR("Hallo, "),7);                     (* schreiben *)
 ok:=Write(MeinHandle,ADR(Name),SIZE(Name));
END NameDemo.
```

Und noch ein Befehl: OPEN. OPEN (er-)öffnet eine Datei und liefert einen Handle auf dieselbe zurück:

FileHandle := Open (Name , Modus);

Name	Zeiger auf den Dateinamen (ADDRESS).
FileHandle	FileHandlePtr-Dateizeiger.
Modus	Art und Weise, wie geöffnet werden soll. Als Parameter sind hier die Werte 1005 und 1006 zulässig, im Definitionsmodul werden sie

auch als die Konstanten oldFile und newFile definiert. Die Konstante oldFile eröffnet eine bereits vorhandene Datei zum Schreib- oder Lesezugriff, wo hingegen newFile eine neue Datei des angegebenen Namens erstellt. Sollte »FileHandle«NIL sein, dann ist irgendein Fehler aufgetreten. Mehr dazu am Ende des DOS-Kapitels.

Geben Sie als Dateinamen »PRT:« und als Modus »oldFile« (oder 1005) an, so können Sie mit WRITE Daten ausdrucken. Das ist die einfache Alternative zur Druckeransteuerung, wie wir sie später noch kennenlernen werden.

Man kann mit OPEN übrigens nicht nur Dateien und Druckerkanäle öffnen, sondern auch Fenster. Allerdings handelt es sich dabei um Konsolenfenster wie das des CLI, also mit Cursor und ohne Schließsymbol. Diese Konsolenfenster eignen sich hervorragend als Eingabefenster! Geben Sie also einen Dateinamen nach folgendem Muster an:

```
CON:x1/y1/Breite/Höhe/"Fenstername"
```

»x1« und »y1« sind die Koordinaten der linken, oberen Ecke des Fensters, »Breite« und »Höhe« erklären sich wohl selbst. »Fenstername« ist der Text, der in der Titelzeile Ihres Fensters stehen soll. So können Sie mit

MeinHandle:=Open(ADR(«CON:0/0/639/255/Gut, was?«),newFile);

ein Konsolenfenster, das sich über den gesamten Bildschirm erstreckt und »Gut, was?« in der Kopfleiste stehen hat, kreieren. Wenn Sie den Dateizeiger »MeinHandle« bei READ- und/oder WRITE-Befehlen verwenden, wirken diese in Ihrem Fenster. Hier liegt der gewaltige Unterschied zu WRITELN & Co.: man ist nicht nur auf das CLI-Fenster beschränkt.

Zum besseren Verständnis noch ein Beispielprogramm:

```
MODULE ConFensterDemo;

FROM SYSTEM IMPORT ADR;
FROM Dos    IMPORT newFile,Close,Delay,Open,Write,FileHandlePtr;

VAR MeinHandle: FileHandlePtr;
   ok         : LONGINT;
  lf          : CHAR;

BEGIN
 lf:=12C;
 MeinHandle:=Open(ADR(«CON:0/0/639/100/Das Fenster"),newFile);(* öffnen  *)
 ok:=Write(MeinHandle,ADR("Das Fenster zum DOS"),19);          (* schreiben *)
```

```
 ok:=Write(MeinHandle,ADR(lf),1);
 Delay(250);                       (* warten  *)
 Close(MeinHandle);                (* schließen *)
END ConFensterDemo.
```

Übrigens: ist Ihnen aufgefallen, daß man in »CON:«-Fenstern z.B. die Cursortasten nicht benutzen kann? Da dies bei manchen Anwendungen störend ist, haben sich die Entwickler des Betriebssystems eine Abhilfe einfallen lassen, das »RAW:«-Fenster nämlich. Ersetzen Sie doch im obigen Beispielprogramm das »CON:« durch ein »RAW:« und betätigen Sie die Cursortasten; im »RAW«-Modus erhalten Sie die Keycodes unbearbeitet (roh eben), was bedeutet, das Sie je nach Anwendung die Nachbearbeitung der Keycodes selber machen müssen oder gezielt diese Möglichkeit nutzen wollen. Außerdem können Benutzer der Workbench 1.3 auch noch das neue Console-Device »NEWCON:« benutzen.

Noch etwas: Wenn Sie nur eine ganz bestimmte Zeit auf einen Tastendruck in einem Konsolenfenster warten wollen (sei es CON: oder RAW:), bedienen Sie sich am besten des WAITFORCHAR-Befehls:

Status := WaitForChar (Handle , Zeit);

Status	ist gleich 0, wenn keine Taste innerhalb des angegebenen Zeitraums gedrückt wurde, andernfalls ist »Status« 1.
Handle	FileHandlePtr auf den Datenstrom oder die Konsole, von dem die Eingabe erfolgen soll.
Zeit	zu wartende Zeit in1/50stel Sekunden.

Sollten Sie mit OPEN eine Datei eröffnet haben, so können Sie mit SEEK den Dateizeiger auf ein beliebiges Zeichen innerhalb Ihres Files setzen:

Position := Seek (Handle , Offset , Modus);

Position	LONGINT-Wert, in dem die neue Position gespeichert ist.
Handle	Zeiger auf die Datei, auf die SEEK angewendet werden soll.
Modus	(LONGINT) bestimmt, ob der in »Offset« (ebenfalls LONGIN) gespeicherte Wert den internen Dateizeiger relativ zur aktuellen Position, zum Dateiende oder -anfang hin verschiebt. Drei Werte sind für »Modus« zulässig: 1. beginning (#–1): verschiebt den Zeiger zum Dateianfang hin, 2. current (#0): verschiebt den Zeiger relativ zur aktuellen Position und 3. end (#1): verschiebt den Zeiger zum Dateiende hin.

Gibt man für »Offset« negative Werte an, so kann man den Dateizeiger auch rückwärts verschieben.

Mit OPEN geöffnete Dateien, Druckerkanäle und Fenster sollten spätestens am Programmende immer durch den Befehl

CLOSE (MeinHandle);

geschlossen werden, da das AmigaDOS ungenutzte FileHandles nicht selbstständig schließen kann. MeinHandle bedeutet: FileHandlePtr auf die zu schließende Datei. Im allgemeinen ist es einfach guter Stil, auf einem Multitasking-Rechner bei der Beendigung des Programms sauber aufzuräumen, d.h. alle angeforderten Resourcen wieder freizugeben.

2.3 Locks

Locks (dt. Schlösser) sind im Prinzip nichts anderes als Dateipfade, wie man sie im CLI verwendet, z.B. »df0:Modules/Libraries/sym«, mit dem Unterschied, daß Locks vom Programm aus verwaltet werden und einige Zusatzmöglichkeiten bieten. Ein Lock muß vom Typ FileLockPtr sein, der unglücklicherweise ein sogenannter BPTR ist, ein Pointer im BCPL-Format, der immer auf den Anfang eines 32-Bit Wortes zeigt, also sozusagen ein normaler Pointer modulo 4 ist. Der Grund für das Existieren der BPTR begründet sich dadurch, daß das Filing System (Verwaltung der Dateien auf einer Disk) des AmigaDOS in der Sprache BCPL geschrieben wurde, die nur den einzigen Datentyp Langwort (32-Bit) und nur Pointer auf diese Langworte kennt. Zum Glück reicht für unsere Zwecke der FileLockPtr, so wie er definiert ist, völlig aus. Durch den Befehl LOCK können Sie Ihren Lock initialisieren. Die Syntax lautet:

MeinLock := Lock (Pfad , Zugriffsmodus);

MeinLock	zu initialisierendes Lock des Typs FileLockPtr.
Pfad	ADDRESS-Wert, muß einen Zeiger auf den String des Dateipfades enthalten.
Zugriffsmodus	entweder –1 (sharedLock) oder –2 (exclusiveLock). Hat dieser Parameter den Wert –1, heißt das, da auch andere Programme auf das Verzeichnis, welches durch das Lock definiert wird, zugreifen können, andernfalls (–2) ist es ausschließlich dem laufenden Programm vorbehalten. Daher muß ein Lock am Programmende unbedingt durch *UNLOCK (MeinLock);*

aufgelöst werden; MeinLock: FileLockPtr auf das zu löschende Lock. Andernfalls können bis zu einem Reset keine Programme mehr auf das entsprechende Verzeichnis zugreifen.

2.3.1 Lock-Befehle

Mit dem Befehl ALTERLOCK machen Sie ein angegebenes Verzeichnis zum Hauptverzeichnis, wie beim CLI-Befehl CD:

AlterLock := CurrentDir (MeinLock);

MeinLock	FileLockPtr auf das zum Hauptverzeichnis zu setzende Lock.
AlterLock	FileLockPtr auf das vorherige Hauptverzeichnis.

Um das Arbeiten mit Locks näherzubringen, folgt jetzt ein Beispielprogramm:

```
MODULE LockDemo;

FROM SYSTEM IMPORT ADR;
FROM Dos    IMPORT FileLockPtr,Lock,CurrentDir,Execute,UnLock;

VAR MeinLock: FileLockPtr;
  ok        : LONGINT;

BEGIN
 MeinLock:=Lock(ADR("SYS:c"),-2);
 MeinLock:=CurrentDir(MeinLock);
 ok:=Execute(ADR("dir"),NIL,NIL);
 UnLock(MeinLock);
END LockDemo.
```

Der im Programm auftauchende Befehl EXECUTE muß noch erklärt werden. Er führt einen ganz normalen CLI-Befehl aus; seine Syntax lautet:

ok := Execute (Befehl , Eingabe , Ausgabe);

ok	LONGINT-Wert, der, bei korrekt ausgeführtem Befehl, 0 oder andernfalls eine Fehlermeldung enthält.
Befehl	(ADDRESS) zeigt auf den String mit dem auszuführenden Befehl.
Eingabe und Ausgabe	FileHandlePtr, die, sollte von einer Datei gelesen oder in eine geschrieben werden, auf diese zeigen. Bei Ein- und Ausgabe vom und auf den Bildschirm sind diese beiden Werte NIL.

Wichtig: Soll ein mit EXECUTE aufgerufener CLI-Befehl ordnungsgemäß durchgeführt werden, so muß im selben Verzeichnis der Befehl RUN stehen!

Weitere Lock-Befehle sind z.B. PARENTDIR, der das übergeordnete Verzeichnis des Locks zurückgibt:

NeuerLock := ParentDir (MeinLock);

MeinLock	FileLockPtr auf das Verzeichnis, dessen übergeordnetes zurückgegeben werden soll.
NeuerLock	FileLockPtr auf das übergeordnete Verzeichnis von »MeinLock«, ebenfalls ein FileLockPtr.

Manchmal möchte man auch von einem Programm aus neue Verzeichnisse erstellen. Mit CREATEDIR ist auch das kein Problem:

MeinLock := CreateDir (Name);

MeinLock	FileLockPtr auf das Lock des neuen Verzeichnisses.
Name	ADDRESS-Wert, zeigt auf einen String mit dessen Namen.

Möchte man ein Lock kopieren, so kann man dies mit DUPLOCK tun:

LockCopy := DupLock (MeinLock);

MeinLock	FileLockPtr auf das zu kopierende Lock.
LockCopy	FileLockPtr-Kopie von »MeinLock«.

Außerdem kann man Dateien (seien es Verzeichnisse oder Programme) untersuchen, indem man sie als Locks definiert und auf diese dann den EXAMINE-Befehl ansetzt:

ok: = Examine (MeinLock , MeinInfoBlock);

ok	LONGINT-Variable, die entweder 0 oder eine Fehlermeldung enthält.
MeinLock	FileLockPtr auf das betreffende Lock.
MeinInfoBlock	FileInfoBlockPtr auf eine Datenstruktur namens FileInfoBlock. Diese Datenstruktur ist, da sie mehrere Daten auf einmal enthält (wie der Name schon sagt), als RECORD definiert:

```
FileInfoBlock=RECORD
 diskKey: LONGINT                ; Diskettennummer
 dirEntryType: LONGINT           ; > 0: Verzeichnis, <= 0: Datei
 fileName: ARRAY[0..107] OF CHAR ; Dateiname
 protection : LONGSET            ; Protectionbits:
                                   0=nicht löschbar,
                                   1=nicht ausführbar,
                                   2=schreibgeschützt,
                                   3=lesegeschützt und
                                   4=archievbit
 entryType: LONGINT              ; ???
 size: LONGINT                   ; Größe in Bytes
 numBlocks: LONGINT; Größe in Blocks
 date    : Date                  ; Datumsstruktur
 comment   : ARRAY[0..115] OF CHAR; Kommentarzeile
END;
```

Die einzelnen Felder wurden schon erklärt, anzumerken ist wohl nur noch, daß der Sinn des Feldes »entryType« im Dunkeln liegt (auch das ROM-KERENEL-MANUAL hüllt sich darüber in Schweigen) und daß die Datumsstruktur später erklärt wird.

Dem FileInfoBlock muß allerdings ein bereits freier Speicherbereich zugewiesen werden, was man mit der Standardprozedur ALLOCATE leider nicht machen kann. So ist die EXEC-Funktion ALLOCMEM zu verwenden, auf die ich hier nicht näher eingehen möchte, da sie im EXEC-Kapitel genauestens erklärt wird. Bitte schlagen Sie bei Bedarf dort nach. Nun folgt ein Beispielprogramm:

```
MODULE ExamineDemo;

FROM SYSTEM IMPORT ADR;
FROM InOut  IMPORT WriteString,WriteInt,WriteLn;
FROM Dos    IMPORT FileInfoBlockPtr,FileLockPtr,Lock,Examine,UnLock;
FROM Exec   IMPORT AllocMem,MemReqs,MemReqSet,FreeMem;

VAR MeinLock    : FileLockPtr;
  MeinInfoBlock: FileInfoBlockPtr;
  ok           : LONGINT;

BEGIN
 MeinInfoBlock:=AllocMem(SIZE(MeinInfoBlock^),MemReqSet{memClear});

 MeinLock:=Lock(ADR("df0:c/dir"),-2);

 ok:=Examine(MeinLock,MeinInfoBlock);

 UnLock(MeinLock);

 WriteLn;
```

```
WriteString("Dateigröße in Bytes :");
WriteInt(MeinInfoBlock^.size,6);
WriteLn;

WriteString("Dateigröße in Blocks:");
WriteInt(MeinInfoBlock^.numBlocks,4);
WriteLn;

WriteString("Kommentarzeile   :");
WriteString(MeinInfoBlock^.comment);
WriteLn;

FreeMem(MeinInfoBlock,SIZE(MeinInfoBlock^));
END ExamineDemo.
```

Es gibt einen dem EXAMINE-Kommando ähnlichen Befehl: INFO. Der Unterschied ist allerdings, daß Sie nicht Informationen über eine Datei, sondern gleich über die ganze Diskette erhalten, wie beim gleichnamigen CLI-Kommando:

ok := Info (MeinLock , MeinInfoDatenBlock);

ok — wie schon so oft, der Rückgabewert der Funktion, entweder 0 oder eine Fehlernummer vom Typ LONGINT.

MeinLock — FileLockPtr, der auf das Lock der Diskette zeigt (lediglich die Laufwerksangabe ist hier relevant).

MeinInfoDatenBlock — InfoDataPtr, der auf einen Record namens InfoData zeigt:

```
InfoData = RECORD
 numSoftErrors: LONGINT ; Anzahl der Software-Errors
                          auf der Disk
 unitNumber   : LONGINT ; Nummer der Diskettenstation
 diskState    : LONGINT ; Diskstatus, siehe unten
 numBlocks    : LONGINT ; Anzahl der Blöcke auf Disk
 numBlocksUsed: LONGINT ; Anzahl der belegten Blöcke
 bytesPerBlock: LONGINT ; Anzahl der Bytes pro Block
 diskType     : LONGINT ; Typ der Diskette, s.u.
 volumeNode   : LONGINT ; Zeiger auf den Diskettennamen
 inUse        : LONGINT ; <> 0, wenn auf Disk
                          zugegriffen wird
END;
```

Die meisten Felder erklären sich wohl von selbst, zu einigen sind noch zusätzliche Anmerkungen nötig:

diskState Dieses Feld kann drei Werte, die im Headerfile DOS als Konstanten definiert sind, enthalten, nämlich:

writeProtect (#80):	Diskette ist schreibgeschützt
validating (#81):	Diskette wird gerade repariert, oder hat einen Fehler, bei dem sich das DOS nicht zu einem Softerror entschließen kann.
validated (#82):	Diskette ist OK. Dieser Wert sollte normal sein.

disktype enthält Informationen über den Diskettentyp:

noDiskPresent (# –1):	Keine Disk im Laufwerk.
unreadableDisk (BAD):	Disk unlesbar (möglicherweise spezielles Aufzeichnungsformat oder unformatiert)
dosDisk (DOS):	Normale DOS-Disk. Dieser Wert sollte hier normal sein.
notReallyDos (NDOS):	Zwar Amiga-Aufzeichnungsformat, aber keine DOS-Disk. Dieser Feldinhalt ist äußerst selten (zum Beispiel benutzt das Festplatten-Backupprogramm Quarterback ein AmigaDOS ähnliches Format).
kickstartDisk (KICK):	Kickstart-Diskette des Amiga 1000.

Es bleibt noch zu sagen, daß ich ein Programm mit dem INFO-Kommando seltsamerweise nur zum Laufen bekommen habe, als ich die DOS-Bibliothek »von Hand« öffnete. Der Grund dafür ist mir allerdings schleierhaft. Auch zum INFO-Befehl folgt ein Demoprogramm:

```
MODULE InfoDemo;

FROM SYSTEM IMPORT ADR;
FROM InOut  IMPORT WriteString,WriteInt,WriteLn;
FROM Dos    IMPORT InfoDataPtr,FileLockPtr,Lock,Info,UnLock;
FROM Exec   IMPORT AllocMem,MemReqs,MemReqSet,LibraryPtr,OpenLibrary;

VAR MeinLock      : FileLockPtr;
  InfoDatenBlock: InfoDataPtr;
  ok            : LONGINT;
  DosBasePtr    : LibraryPtr;

BEGIN
  DosBasePtr:=OpenLibrary(ADR("dos.library"),0);
```

```
 InfoDatenBlock:=AllocMem(SIZE(InfoDatenBlock),MemReqSet{memClear}); (* Schon mit-
 Nullen gefüllten Speicher  anfordern *)

 MeinLock:=Lock(ADR("df0:"),-2); (* Lock definieren *)

 ok:=Info(MeinLock,InfoDatenBlock);

 UnLock(MeinLock);

 WriteLn;

 WriteString("Errors auf Diskette:");
 WriteInt(InfoDatenBlock^.numSoftErrors,2);
 WriteLn;

 WriteString("Diskstatus  :");
 WriteInt(InfoDatenBlock^.diskState,2);
 WriteLn;

 WriteString("Blocks auf Disk  :");
 WriteInt(InfoDatenBlock^.numBlocks,4);
 WriteLn;

 WriteString("Benutzte Blöcke  :");
 WriteInt(InfoDatenBlock^.numBlocksUsed,4);
 WriteLn;

 WriteString("Bytes per Block  :");
 WriteInt(InfoDatenBlock^.bytesPerBlock,3);
 WriteLn;

END InfoDemo.
```

2.4 Weitere Befehle

Besonders zur Directoryanalyse ist der Befehl EXNEXT geeignet. Sollten Sie mit EXAMINE ein Verzeichnis untersuchen, so können Sie mit EXNEXT den Info-Datenblock mit den Daten des nächsten Eintrags im Directory füllen.

ok := ExNext (MeinLock , MeinInfoBlock);

Die Parameter entsprechen denen von EXAMINE, lediglich »MeinInfoBlock« muß bereits mit Daten von EXAMINE gefüllt sein. Will man Dateien vom Programm aus löschen, kann man sich der Kommandos RENAME und DELETEFILE bedienen. Die Syntax von RENAME lautet:

ok := Rename (alterName , neuerName);	
ok	LONGINT-Rückgabewert (0 für OK, sonst eine Fehlermeldung).
alterName	ADDRESS-Zeiger auf den Namen der umzubenennenden Datei.
neuerName	ADDRESS-Zeiger auf den neuen Namen der Datei.

DELETEFILE hat folgende Form:

ok := DeleteFile (Name);	
ok	LONGINT-Wert für die Rückmeldung.
Name	ADDRESS-Zeiger auf den Namen der zu löschenden Datei.

Kommentare, wie sie im Feld »comment« des FileInfoBlocks stehen, können mit SET-COMMENT erzeugt werden:

ok := SetComment (Dateiname , Kommentar);	
ok	LONGINT-Rückmeldung.
Dateiname	ADDRESS-Zeiger auf den Namen der zu kommentierenden Datei.
Kommentar	ADRESSE-Wert, der die Adresse eines Strings mit maximal 80 Zeichen für den Kommentar angibt.

Auch zum CLI-Befehl PROTECT, der Dateien schützt, gibt es ein äquivalentes Kommando, SETPROTECT:

ok := SetProtect (Dateiname , Schutzmuster);	
ok	LONGINT-Rückmeldung.
Dateiname	ADDRESS-Zeiger auf den Namen der zu schützenden Datei.
Schutzmuster	LONGSET-Wert, der die Art des Schutzes in einerBitkombination enthält: Bit 0 gesetzt: Datei ist nicht löschbar Bit 1 gesetzt: Datei ist nicht ausführbar Bit 2 gesetzt: Datei ist schreibgeschützt Bit 3 gesetzt: Datei ist lesegeschützt Bit 4 gesetzt: Archievbit

Manchmal möchte man ein Programm einfach beenden und dem CLI eine Fehlermeldung zurückgeben. Dafür ist der Befehl EXIT zuständig. Er wird von C-Programmierern oft gebraucht, um das Programm abzubrechen, wenn z.B. eine Bibliothek nicht eröffnet werden konnte. Mit dem M2-Compiler geht das aber dank der Prozedur TERMPROCEDURE aus Arts besser und eleganter. Der Vollständigkeit halber sei der EXIT-Befehl trotzdem erwähnt:

Exit (Fehlermeldung);

Fehlermeldung	LONGINT-Wert, der dem CLI zurückgegeben wird. Es sind im DOS-Headerfile drei Konstanten für die EXIT-Fehlermeldung definiert: ok (# 0) : für fehlerfreien Abbruch (?!) warn (# 5) : für einen leichten Fehler (Warnung) error (# 10): für einen schweren Fehler (Error) fail (# 20) : für einen fatalen Fehler (Fail)

Um die aktuelle Systemzeit abzufragen, gibt es den Befehl DATESTAMP:

DateStamp (Datumszeiger);

Datumszeiger — DatePtr auf den Date-Record, in den die Systemzeit hineingeschrieben werden soll:

```
Date = RECORD
  days :  LONGINT ; vergangene Tage nach dem 1.1.78
  minute: LONGINT ; vergangene Minuten seit Mitternacht
  tick :  LONGINT ; nach obiger Minute vergangene
                    1/50stel Sekunden (IntuiTicks)
END;
```

Sollte die Systemzeit nicht gesetzt sein, erhalten alle drei Felder den Wert 0.

Mit ISINTERACTIVE kann man prüfen, ob der angegebene Handle ein Zeiger auf ein virtuelles Terminal ist, über das man Informationen mit dem Programm austauschen kann:

Status := IsInteractive (Handle);

Handle	FileHandlePtr-Handle des zu prüfenden Kanals.
Status	Rückgabewert, ist gleich –1, wenn »Handle« ein Dateizeiger auf ein virtuelles Terminal, z.B. ein Konsolenfenster ist oder ein anderer

	Wert, wenn »Handle« kein solcher ist. Gibt man beispielsweise den Rückwert von OUTPUT an, so enthält »Status« ganz sicher –1.

Sollen Programmsegmente eingeladen werden, so kann man sich des LOADSEG-Befehls bedienen:

Segment := LoadSeg (Name);

Name	ADDRESSe auf den Namen des zu ladenden Segments.
Segment	ADDRESSe des eingeladenen Segments. Ist sie NIL, trat irgendein Fehler beim Ladevorgang auf.

Eines mit LOADSEG geladenen Programmsegments wird man mit UNLOADSEG wieder ledig:

UnLoadSeg (Segment);

Segment	ADDRESSe des Segments, das entfernt werden soll.

Um einen neuen Prozeß zu erstellen, benötigt man die CREATEPROC-Funktion:

Prozeß := CreateProc (Name , Priorität , Segment , Stack);

Name	ADDRESS-Zeiger auf den Namen des Prozesses, der generiert werden soll.
Priorität	LONGINT-Priorität des neuen Prozesses.
Segment	ADDRESS-Zeiger auf das erste Segment der Liste, in der der Code des zu startenden Programmes stehen soll.
Stack	LONGINT-Größe des Stacks in Bytes.
Prozeß	ProcessPtr auf den Process-Record:

```
Process=RECORD
 task            : Task      ; Taskstruktur des Prozesses
                               (siehe EXEC-Kapitel)
 msgPort         : MsgPort   ; Nachrichtenportstruktur
                               des Prozesses
                               (siehe EXEC-Kapitel)
 pad             : WORD      ; fürs System
 segList         : ADDRESS   ; Adresse der Segmentliste
 stackSize       : LONGINT   ; Stackgröße
```

```
        globVec          : ADDRESS     ; globale Vektoren
        taskNum          : LONGINT     ; Nummer des CLI-Tasks oder
                                         0, wenn nicht vom CLI
                                         gestartet
        stackBase        : ADDRESS     ; Zeiger auf Stackbasis
        result2          : LONGINT     ; Resultat vom letzten Aufruf
        currentDir       : FileLockPtr; Lock aufs aktuelle
                                         Directory
        cis              : ADDRESS     ; Adresse aufs Handle des
                                         Eingabestroms
        cos              : ADDRESS     ; Adresse aufs Handle des
                                         Ausgabestroms
        consoleTask      : MsgPortPtr ; Konsolenhandler des
                                         aktuellen Windows
        fileSystemTask   : MsgPortPtr ; Dateihandler des aktuellen
                                         Laufwerks
        cli              : ADDRESS     ; Adresse des aktuellen CLIs
        returnAddr       : ADDRESS     ; Zeiger zum voherigen Stack
        pktWait          : ADDRESS     ; aufzurufende Funktion wenn
                                         Nachricht eintrifft
        windowPtr        : ADDRESS     ; Zeiger auf das Fenster, in
                                         dem die Fehlermeldungen
                                         ausgegeben werden
      END;
```

Um einen Prozeß, der einen Ein-/Ausgabekanal verwendet, zu identifizieren, gibt es den DEVICEPROC-Befehl:

Prozeß := DeviceProc (Name);

Prozeß	ProcessPtr auf den zu identifizierenden Prozeß.
Name	Zeiger auf den Namen des Ein-/Ausgabekanals, dessen Prozeß ermittelt werden soll. Ist »Prozeß« NIL, so trat ein Fehler auf.

2.5 Fehlermeldungen

In den voherigen Kapiteln wurde bei der Befehlssyntax oft geschrieben, daß »irgendein« Fehler aufgetreten sein muß, wenn der Rückgabewert 0 oder –1 enthält. Doch was für ein Fehler? Mit IOERR läßt er sich präzisieren:

Fehlernummer := IoErr();	
Fehlernummer	LONGINT-Wert, der die Nummer des letzten aufgetretenen Fehlers enthält. Sollte kein Fehler aufgetreten sein, enthält »Fehlernummer« 0.

Es gibt folgende Fehlermeldungen (Kickstart v1.2):

noFreeStore (# 103):

Es ist nicht genügend Speicherplatz frei.

taskTableFull (# 104):

Es sind bereits 20 Prozesse aktiv.

lineTooLong (# 120):

Die Argumente für diesen Befehl sind entweder nicht korrekt oder es gibt zuviele.

fileNotObject (# 121):

Die aufgerufene Datei ist kein lauffähiges Programm.

invalidResidentLibrary (# 122):

Die aufgerufene Resident-Bibliothek ist ungültig

noDefaultDir (# 201):

Kein Default-Directory.

objectInUse (# 202):

Die Datei wird momentan von einem anderen Task benutzt.

objectExists (# 203):

Die Datei existiert bereits.

dirNotFound (# 204):

Das Verzeichnis kann nicht gefunden werden.

objectNotFound (# 205):

Das File kann nicht gefunden werden.

badStreamName (# 206):

Unzulässiger Name für Ein-/Ausgabekanal.

objectTooLarge (# 207):

Die Datei ist zu groß.

actionNotKnown (# 209):
Der Anforderungscode ist ungültig.

invalidComponentName (# 210):
Der Dateiname ist ungültig.

invalidLock (# 211):
Das Lock ist ungültig.

objectWrongType (# 212):
Der Objekttyp ist ungültig.

diskNotValidated (# 213):
Die Diskette ist ungültig.

diskWriteProtected (# 214):
Die Diskette ist schreibgeschützt.

renameAcrossDevices (# 215):
Die Datei kann nicht über verschiedene Geräte (z.B. von DF0: nach DF1:) umbenannt werden.

directoryNotEmpty (# 216):
Das Verzeichnis enthält Dateien und kann nicht gelöscht werden.

tooManyLevels (# 217):
Mehr als 256 Unterverzeichnisse in einem Verzeichnis gefunden.

deviceNotMounted (# 218):
Die angesprochene Diskette liegt nicht im Laufwerk.

seekError (# 219):
Es trat ein Fehler beim SEEK-Befehl auf (ungültige Argumente).

commentTooBig (# 220):
Die Kommentarzeile ist länger als 80 Zeichen.

diskFull (# 221):
Die Diskette ist voll.

deleteProtected (# 222):
Es wurde versucht, eine geschützte Datei zu löschen.

writeProtected (# 223):
Es wurde versucht, eine geschützte Datei zu überschreiben.

readProtected (# 224):
Es wurde versucht, eine geschützte Datei zu lesen.

notADosDisk (# 225):
Die Diskette im Laufwerk entspricht nicht dem DOS-Format.

noDisk (# 226):
Es liegt keine Diskette im Laufwerk.

noMoreEntries (# 227):
Es sind keine Verzeichniseinträge mehr vorhanden (bei EXNEXT).

Die hier aufgeführten Fehler entstammen dem DOS-Definitionsmodul. Wenn Sie eine ausführliche Fehlerbeschreibung sowie Abhilfevorschläge suchen, so sehen Sie bitte im DOS-Handbuch zum Amiga nach.

Kapitel 3

Intuition

Intuition ist die Benutzeroberfläche des Amiga. Sie steuert alles, was mit Pull-down-Menüs, Maussteuerung, Auswahlfeldern usw. zu tun hat. Deluxe-Paint, die Workbench, Sonix und viele andere Programme sind voll in Intuition eingebunden, denn bei ihnen kann man die meisten und wichtigsten Operationen mit der Maus durchführen. In diesem Kapitel soll erklärt werden, wie man Programme in Intuition einbindet, und das ist in den meisten Programmiersprachen ziemlich ähnlich. Sollten Sie C oder Maschinensprache beherrschen, wird Ihnen hier vieles bekannt vorkommen.

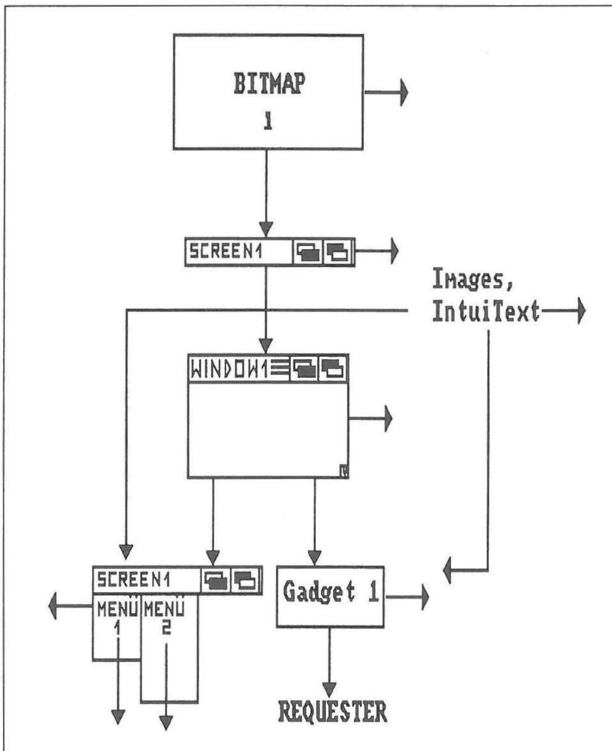


Bild 3.1: Eine Auswahl der Intuition-Elemente

3.1 Die Programmierung

Intuition verwaltet fast alles selbst, der Programmierer muß dem Betriebssystem nur seine Wünsche in Form von Records mitteilen. Die relevanten Records und Befehle werden im folgenden besprochen. Es ist für die folgenden Kapitel wichtig zu wissen, daß Intuition fast alle Daten intern mit Listen verwaltet. So kommen in vielen Strukturen Zeiger auf ein folgendes Fenster, Gadget, Menü oder was auch immer vor.

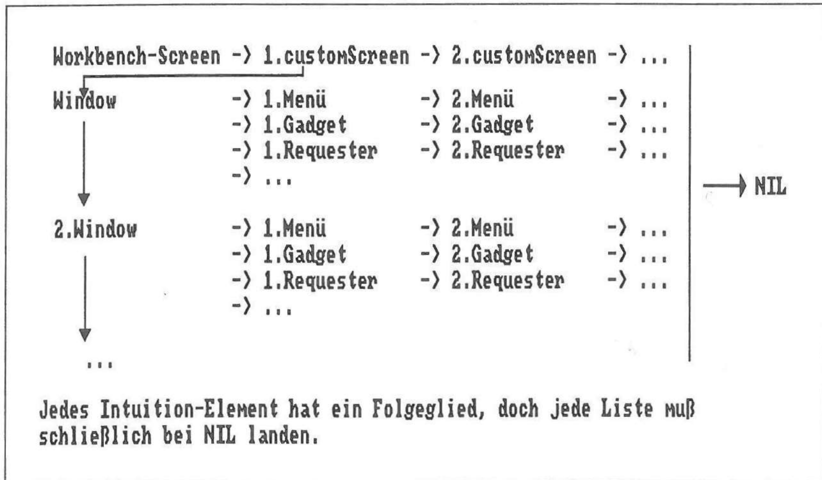


Bild 3.2: So funktioniert das interne Listensystem

3.2 Screens

Da der Amiga ein Multitasking-Computer ist, viele Programme den Bildschirm für sich beanspruchen und anderen keinen Platz lassen, aber kaum jemand sich drei, vier oder mehr Monitore kaufen würde, mußte ein Kompromiß her, virtuelle Bildschirme (Screens) genannt, die sich vertikal verschieben lassen und mehrfach überlappen dürfen. Wir wollen nun unsere eigenen konstruieren.

3.2.1 Die NewScreen-Struktur

Um einen Screen individuell zu gestalten, kann man die wichtigsten Parameter in einem Record angeben, dem NEWSCREEN-Record:

```
NewScreen=RECORD
  leftEdge    : INTEGER       ; x-Koordinate der linken, oberen Ecke
  topEdge     : INTEGER       ; y-Koordinate der linken, oberen Ecke
```

```
  width        : INTEGER       ; Breite des Screens
  height       : INTEGER       ; Höhe des Screens
  depth        : INTEGER       ; "Tiefe" des Screens in Bitplanes
  detailPen    : UByte         ; Vordergrundfarbe
  blockPen     : UByte         ; Hintergrundfarbe
  viewModes    : ViewModeSet   ; Darstellungsmodi
  type         : ScreenFlagSet ; Screenflags
  font         : TextAttrPtr   ; Zeiger auf den Zeichensatz (vorerst wollen wir
                                 NIL verwenden (Standard))
  defaultTitle: ADDRESS        ; Zeiger auf den Titelstring
  gadgets      : GadgetPtr     ; Zeiger auf die Gadgetstruktur (vorerst NIL)
  customBitMap: BitMapPtr      ; Zeiger auf eigene Bitmap, vorerst bitte NIL
END;
```

Einige Felder dürften bereits klar sein, die anderen sollen nun erläutert werden:

depth gibt die Tiefe eines Screens in Bitplanes an. Ein Screen kann maximal 2^depth Farben darstellen:

Farben	Tiefe
2	1
4	2
8	3
16	4
32	5
64	6 (Halfbrite, siehe GRAPHICS-Kapitel)
4096	6 (HAM, siehe GRAPHICS-Kapitel)

Beachten Sie bitte, daß Sie im Normalfall, der für das gesamte Intuition-Kapitel gilt, nur eine Tiefe von maximal 5 Bitplanes angeben sollten.

detailPen Dieses Feld enthält die Nummer des Farbregisters, in dessen Farbe Text, Gadgets und die Titelleiste des Screens gezeichnet werden sollen.

blockPen Hier steht die Farbe für den Texthintergrund.
Beachten Sie bitte, daß der Screenhintergrund stets mit der Farbe aus Register 0 gezeichnet wird (beim Workbench-Screen ursprünglich blau). Die einzelnen Pens enthalten beim Öffnen eines Screens folgende Farbwerte:

	Nummer	Farbe
bei 1 Bitplane	0	blau (Hintergrundfarbe)
	1	weiß
bei 2 Bitplanes	2	schwarz
	3	orange
bei 3 Bitplanes	4	blau
	5	violett
	6	türkis
	7	weiß
bei 4 Bitplanes	8	schwarz
	9	rot
	10	grün
	11	braun
	12	blau
	13	blau
	14	grün
	15	grau

viewModes Die Menge viewModes gibt den Darstellungsmodus des Screens an. Folgende Angaben sind zulässig:

genlocVideo Für Mischung von Video- und Computerbildern

lace Verdopplung der vertikalen Zeilen (von normal 256 (PAL)oder 200 (NTSC) auf 512 bzw. 400), muß, je nach Monitor, mit mehr oder minder starkem Flackern erkauft werden.

pfba Legt Prioritäten für Double-Playfield-Modus fest, der im GRAPHICS-Kapitel besprochen wird.

extraHalfbrite 64 Farben gleichzeitig darstellbar (folgt im GRAPHICS-Kapitel).

genlocAudio Wird im Moment noch nicht durch die Systemsoftware/Hardware richtig unterstützt.

dualpf Für Dual-Playfield-Modus. Bitte momentan nicht beachten.

ham 4096 Farben gleichzeitig darstellbar. Näheres im GRAPHICS-Kapitel.

vpHide Ein Screen wird erzeugt, aber nicht dargestellt.

sprites	Ermöglicht die Darstellung von Hardware-Sprites (GRAPHICS).
hires	Muß gesetzt werden, wenn mit 640*256 Punkten Auflösung gearbeitet werden soll (wie beim Workbench-Screen).
screenType	Hier wird der Screentyp bezeichnet. Die verschiedenen Typen sind in der Menge »ScreenFlags« zu finden:
wbenchScreen	Der Workbench-Screen.
customScreen	Eigener Screen.
customBitmap	Eigene Bitmap für den Screen.
screenBehind	Screen wird im Hintergrund geöffnet.
screenQuiet	Der Screen besitzt weder Gadgets noch Titelleiste.
showTitle	Wird von Intuition gesetzt, wenn die Titelleiste des Screens sichtbar ist.
beeping	Wird von Intuition gesetzt, wenn der Screen aufblinkt (»beept«).
font	In diesem Feld wird ein Zeiger auf den Zeichensatz-Record übergeben. Vorerst wollen wir uns jedoch mit dem Standard-Zeichensatz begnügen.
defaultTitle	Hier wird ein Zeiger auf den Text in der Titelleiste übergeben.
gadgets	Enthält einen Zeiger auf die Struktur eines Gadgets. Bis zum (Unter-)Kapitel über Gadgets ist jedoch NIL anzugeben.
customBitmap	Enthält einen Zeiger auf die Bitmapstruktur unseres Screens. Im Moment soll uns aber die Bitmap, die Intuition unserem Screen beim Öffnen zuteilt, genügen.

3.2.2 Die ersten Screen-Befehle

Und nun ist es an der Zeit, uns den ersten Screen-Befehlen zuzuwenden. Zuerst brauchen wir natürlich einen Befehl, der unseren Screen öffnet. Dafür ist OPENSCREEN gedacht:

ScreenZeiger := OpenScreen(MeinNeuerScreen);

MeinNeuerScreen	NewScreen-Record unseres definierten Screens.
ScreenZeiger	ScreenPtr auf den Screen-Record des geöffneten Screens:

```
Screen=RECORD
  nextScreen    : ScreenPtr    ; Zeiger auf den nächsten Screen der Liste
  firstWindow   : WindowPtr    ; Zeiger auf das erste Fenster des Screens
  leftEdge      : INTEGER      ; x-Koordinate der linken, oberen Ecke des Screens
  topEdge       : INTEGER      ; y-Koordinate der linken, oberen Ecke des Screens
  width         : INTEGER      ; Breite des Screens
  height        : INTEGER      ; Höhe des Screens
  mouseY        : INTEGER      ; y-Koordinate der augenblicklichen Mausposition
  mouseX        : INTEGER      ; x-Koordinate der augenblicklichen Mausposition
  flags         : ScreenFlagSet ; Screen-Flags
  title         : ADDRESS      ; Zeiger auf augenblicklichen Text in der
                                 Titelzeile
  defaultTitle  : ADDRESS      ; Zeiger auf Standard-Text in der Titelzeile
                                 (wie er beim Öffnen übergeben wird)
  barHeight     : Byte         ; Höhe der Titelleiste
  barVBorder    : Byte         ; vertikale Grenze der Titelleiste
  barHBorder    : Byte         ; horizontale Grenze der Titelleiste
  menuVBorder   : Byte         ; vertikale Grenze des Menüs
  menuHBorder   : Byte         ; horizontale Grenze des Menüs
  wBorTop       : Byte         ; Dimension des obereren Fensterrahmens
  wBorLeft      : Byte         ; Dimension des linken Fensterrahmens
  wBorRight     : Byte         ; Dimension des rechten Fensterrahmens
  wBorBottom    : Byte         ; Dimension des unteren Fensterrahmens
  font          : TextAttrPtr  ; Zeiger auf den Zeichensatz-Record
  viewPort      : ViewPort     ; ViewPort-Record (kommt noch ...)
  rastPort      : RastPort     ; RastPort-Record (GRAPHICS-Kapitel)
  bitMap        : BitMap       ; Bitmap-Record
  layerInfo     : LayerInfo    ; Layer-Record (Kern des ganzen Screen- und
                                 Window-Systems)
  firstGadget   : GadgetPtr    ; Zeiger auf das erste Gadget des Screens
  detailPen     : UByte        ; Detail-Farbregister des Screens
  blockPen      : UByte        ; Block-Farbregister des Screens
  saveColorØ    : CARDINAL     ; Sicherheitskopie der Farbpalette fürs Aufblinken
  barLayer      : LayerPtr     ; Zeiger auf Screen-Layer
  extData       : ADDRESS      ; wird in Zukunft ab der Betriebsystemversion
                                 1.4 für zum Beispiel den Hedleymonitor benutzt.
  userData      : ADDRESS      ; Hier kann der Benutzer einen Zeiger auf eigene
                                 Datenstrukturen anbringen, um weitere Daten für
                                 den Screen bereitzuhalten. Das ist eine sehr
                                 fortgeschrittene und selten genutzte Technik.
END;
```

Die Felder Viewport, Rastport oder Bitmap werden im GRAPHICS-Kapitel eingehend besprochen, doch sehen Sie schon, daß man mit diesem Record ganz interessante Sachen, wie das Abfragen der Mauskoordinaten, machen kann. Ist »ScreenZeiger« NIL, so muß etwas schief gelaufen sein.

Um einen Screen zu schließen, muß man den Befehl CLOSESCREEN verwenden:

CloseScreen (ScreenZeiger);

ScreenZeiger	ScreenPtr auf den zu schließenden Screen.

Jetzt will ich Ihnen ein Beispielprogramm zeigen, das einen einfachen Screen öffnet und schließt:

```
MODULE ScreenDemo;

FROM SYSTEM    IMPORT ADR;
FROM Intuition IMPORT NewScreen,ScreenPtr,OpenScreen,CloseScreen,customScreen;
FROM Graphics  IMPORT ViewModes,ViewModeSet;
FROM Dos       IMPORT Delay;

VAR MeinScreen : ScreenPtr;
    ScreenDaten: NewScreen;

BEGIN

  WITH ScreenDaten DO (* Daten initialisieren *)
    leftEdge:=0; topEdge:=0; width:=640; height:=256;
    depth:=2; detailPen:=0; blockPen:=1; viewModes:=ViewModeSet{hires};
    type:=customScreen; font:=NIL; defaultTitle:=ADR("TEST-SCREEN");
    gadgets:=NIL; customBitMap:=NIL;
  END;

  MeinScreen:=OpenScreen(ScreenDaten); (* Öffnen *)

  Delay(250); (* Warten *)

  IF MeinScreen#NIL
    THEN CloseScreen(MeinScreen);
  END; (* Schließen *)

END ScreenDemo.
```

3.2.3 Weitere Screen-Befehle

Um einen Screen hoch und runter zu bewegen, was man normalerweise mit der Maus erledigt, gibt es auch einen Befehl, der dies programmgesteuert tut, MOVESCREEN:

MoveScreen (MeinScreen , x , y);

MeinScreen	ScreenPtr auf den zu bewegenden Screen
x	INTEGER-Wert der Bildpunkte, um die der Screen in x-Richtung verschoben werden soll. Da es momentan nicht möglich ist, einen Screen horizontal zu verschieben, sollten Sie hier 0 angeben.
y	INTEGER-Wert der Bildpunkte, um die der Screen in der Vertikalen verschoben werden soll. Positive Werte scrollen den Screen nach unten, negative hingegen schieben den Screen nach oben. Doch zu diesem Befehl ein Beispielprogramm, das einen Hi-Res- (640*256 Punkte) und einen Lo-Res- (320*256 Punkte) Screen öffnet und diesen verschiebt:

```
MODULE ScreenMoveDemo;

FROM SYSTEM    IMPORT ADR;
FROM Intuition IMPORT NewScreen,ScreenPtr,OpenScreen,CloseScreen,customScreen,
                      MoveScreen;
FROM Graphics  IMPORT ViewModes,ViewModeSet;
FROM Dos IMPORT Delay;

VAR MeinScreen1,MeinScreen2  : ScreenPtr;
    ScreenDaten1,ScreenDaten2: NewScreen;

BEGIN

  WITH ScreenDaten1 DO (* Daten initialisieren *)
    leftEdge:=0; topEdge:=0; width:=640; height:=256;
    depth:=2; detailPen:=0; blockPen:=1; viewModes:=ViewModeSet{hires};
    type:=customScreen; font:=NIL; defaultTitle:=ADR("HiRes-Screen");
    gadgets:=NIL; customBitMap:=NIL;
  END;

  WITH ScreenDaten2 DO
    leftEdge:=0; topEdge:=0; width:=320; height:=512;
    depth:=2; detailPen:=0; blockPen:=1; type:=customScreen; font:=NIL;
    viewModes:=ViewModeSet{lace}; defaultTitle:=ADR("LoRes-Screen");
    gadgets:=NIL; customBitMap:=NIL;
  END;
```

```
  MeinScreen1:=OpenScreen(ScreenDaten1); (* Öffnen *)
  IF MeinScreen1 # NIL THEN
     MeinScreen2:=OpenScreen(ScreenDaten2);
     IF MeinScreen2 # NIL THEN
        MoveScreen(MeinScreen2,0,180);
        Delay(200);
        CloseScreen(MeinScreen2);
     END;
     Delay(200);
     CloseScreen(MeinScreen1);
  END

END ScreenMoveDemo.
```

Wollen Sie einen Screen aufblinken lassen, wie es das Amiga-BASIC tut, wenn es einen Fehler gefunden hat, benutzen Sie DISPLAYBEEP:

DisplayBeep (MeinScreen);

MeinScreen	ScreenPtr auf den Screen, dessen Hintergrundfarbe kurz geändert werden soll (nichts anderes bewirkt ja das Aufblinken). Wird hier NIL angegeben, blinken alle geöffneten Intuition-Screens auf. DISPLAYBEEP wird von Programmen gerne dazu benutzt, den Anwender auf etwas aufmersam zu machen.

Um Screens in den Vordergrund zu holen, bzw. nach hinten zu schicken, bedienen Sie sich ja der beiden Gadgets in der Titelleiste oder <Amiga links> + <M> bzw. <Amiga links> + <N>. Doch auch von einem Programm aus können Sie die Tiefenanordnung von Screens ändern. Die beiden Befehle hierzu lauten:

ScreenToFront (MeinScreen);

MeinScreen	ScreenPtr auf den Screen, der in den Vordergrund geholt werden soll.

ScreenToBack (MeinScreen);

MeinScreen	ScreenPtr auf den Screen, der in den Hintergrund gebracht werden soll.

Was man mit eigenen Screens machen kann, ist auch mit dem Workbench-Screen möglich. Möchten Sie ihn aus Speichermangel oder anderen Gründen schließen, so tun Sie das am besten mit CLOSEWORKBENCH:

ok := CloseWorkBench ();

ok	BOOLEAN-Wert, der TRUE ist, wenn der Workbench-Screen geschlossen werden konnte oder FALSE, wenn das Gegenteil eintrat (möglicherweise weil er schon geschlossen wurde).

Sollte man die Workbench wieder öffnen wollen (am Ende eines Programms, das sie geschlossen hat), so bietet sich hier der OPENWORKBENCH-Befehl an:

Workbench := OpenWorkBench ();

Workbench	ScreenPtr auf den Workbench-Screen. Wird NIL zurückgeliefert, so konnte die Workbench nicht geöffnet werden (weil sie vielleicht schon offen ist). Doch wie sich der Workbench-Screen öffnen und schließen läßt, kann man ihn auch in den Vorder- und Hintergrund bringen. Mit WBENCHTOBACK läßt sich die Workbench nach hinten bringen:

ok := WBenchToBack ();

ok	BOOLEAN-Rückgabewert; ist TRUE, wenn die Workbench in den Hintergrund gebracht werden konnte.

Das Gegenstück dazu ist WBENCHTOFRONT:

ok := WBenchToFront ();

ok	BOOLEAN-Rückgabewert; TRUE bei Erfolg, die Workbench nach vorn zu bringen.

3.3 Windows

Windows (Fenster) sind Elemente auf den Screens. Klicken Sie von der Workbench aus ein Diskettensymbol oder ein Unterverzeichnis an, so erscheint ein Window mit dem Inhalt der Diskette (bzw. des Unterverzeichnisses). Das Window ist ein grundlegendes Element der Programmgestaltung, da fast die ganze Ein-/Ausgabesteuerung über Windows abläuft. Windows haben im Vergleich zu Screens viele Gesichter – Konsolenfenster, die Sie im DOS-Kapitel kennengelernt haben, Windows mit Menüs, Gadgets, Scrollbalken, solche zum Vergrößern, Hin- und Herschieben und so weiter. In diesem Kapitel wollen wir die Programmierung und Behandlung der Windows besprechen.

3.3.1 Ein neues Window

Um ein neues Window zu erstellen, kann man:

1. Sich des Windows-Moduls (M2Amiga; wird hier besprochen), bzw. der »Simplified Amiga Library« (Benchmark Modula-2) bedienen (einfach und problemlos).
2. Per Hand, ähnlich eines Screens, eine Fenster-Struktur ausfüllen (komplizierter, bietet aber riesiges Potential an Möglichkeiten).

Es ist wohl das Beste, beide Wege hintereinander zu besprechen.

3.3.2 Das Windows-Modul

Um ein Window zu öffnen, kann man aus Windows den Befehl OPENWINDOW importieren. Dessen Syntax lautet folgendermaßen:

OpenWindow (MeinWindow , x , y , Breite , Höhe , Titel , Gadgets);

MeinWindow	WindowPtr, der auf die Struktur des neuen Windows zeigen soll (wird durch OPENWINDOW initialisiert)
x	x-Koordinate der linken, oberen Ecke des Windows
y	y-Koordinate der linken, oberen Ecke des Windows
Breite	Breite des Windows
Höhe	Höhe des Windows
Titel	Ein String (kein Zeiger!) mit dem Windowtitel
Gadgets	Gadgets, die das Window erhalten soll aus der Menge WinGad: sizing (Größengadget) moving (Fenster kann verschoben werden) arranging (Tiefen-Gadgets) closing (Schließgadget) scrolling (Scrollbalken, Besonderheit von WINDOWS!)

Das Schließgadget und den Scrollbalken muß der Programmierer allerdings selbst abfragen. Wie das geht, wird später erklärt.

Ohne weiteres Zutun seitens des Programmierers erscheint so ein Window auf dem Workbench-Screen. Da es aber Leute geben soll, die ihre Windows gern auf ihrem eigenen Screen sehen möchten, gibt es den Befehl SETSCREEN. SETSCREEN bestimmt irgendeinen geöffneten Screen zum Untergrund für das kommende Window:

SetScreen (MeinScreen);

MeinScreen	ScreenPtr auf den Screen, der zum Screen für alle nachfolgenden Aufrufe von OPENWINDOW erklärt werden soll. Soll der Workbench-Screen wieder eingesetzt werden, muß »MeinScreen« NIL sein.

Die Erklärungen für die weiteren WINDOWS-Befehle entnehmen Sie bitte Ihrem M2Amiga-Handbuch. Ich denke, die Dokumentation reicht an dieser Stelle (ausnahmsweise) einmal aus.

3.3.3 Windows »per Hand«

Es gibt aber noch die Möglichkeit, Windows durch den OPENWINDOW-Befehl der Intuition-Bibliothek zu öffnen:

MeinWindow := OpenWindow (WindowDaten);

MeinWindow	WindowPtr auf das geöffnete Fenster (ist NIL, wenn ein Fehler auftrat).
WindowDaten	NewWindow-Record, in dem die Daten des Fensters stehen. Dieser Record muß, wie bei der NewScreen-Struktur, vor Aufruf von OPENWINDOW ausgefüllt sein:

```
NewWindow=RECORD
  leftEdge    : INTEGER       ; x-Koordinate der linken, oberen Ecke des Fensters
  topEdge     : INTEGER       ; y-Koordinate der linken, oberen Ecke des Fensters
  width       : INTEGER       ; Breite des Fensters
  height      : INTEGER       ; Höhe des Fensters
  detailPen   : UByte         ; siehe NewScreen-Struktur
  blockPen    : UByte         ; siehe NewScreen-Struktur
  idcmpFlags  : IDCMPFlagSet  ; Nachrichten-Flags (s.u.)
  flags       : WindowFlagSet ; Window-Flags (s.u.)
  firstGadget: GadgetPtr      ; Zeiger auf das erste Gadget im Fenster
  checkMark   : ImagePtr      ; Zeiger auf den Haken, der beim Anklicken mancher
                                Menüpunkte erscheint
  title       : ADDRESS       ; Zeiger auf den String mit dem Windowtitel
  bitMap      : BitMapPtr     ; Zeiger auf eine eigene Bitmap (im Moment sollte
                                Ihnen die Standard-Bitmap genügen)
  minWidth    : INTEGER       ; kleinstmögliche Fensterbreite (beim Vergrößern
                                oder Verkleinern)
```

```
  minHeight  : INTEGER       ; kleinstmögliche Fensterhöhe (beim Vergrößern oder
                               Verkleinern)
  maxWidth   : CARDINAL      ; größtmögliche Fensterbreite (beim Vergrößern oder
                               Verkleinern)
  maxHeight  : CARDINAL      ; größtmögliche Fensterhöhe (beim Vergrößern oder
                               Verkleinern)
  type       : ScreenFlagSet ; Typ des Screens, auf dem das Fenster erscheinen
                               soll (wbenchScreen für Workbench, customScreen
                               für eigenen Screen)
END
```

Zu den noch offenen Feldern:

windowFlags Über dieses Feld spezifiziert man den Typ des Windows näher. Zulässig sind die Werte aus der Menge WindowFlags:

windowSizing Ist dieses Flag gesetzt, ist ein Größengadget am rechten, unteren Fensterrand vorhanden.

windowDrag Durch Setzen dieses Flags kann man angeben, ob ein Window in seiner Position verschiebbar ist.

windowDepth Setzen Sie dieses Flag, sind am rechten, oberen Fensterrand zwei Tiefengadgets vorhanden, über die Sie das Window in den Vorder- oder Hintergrund setzen können.

windowClose Ihr Window verfügt über ein Schließgadget, wenn Sie dieses Flag setzen.

sizeBRight Dieses Flag, das von vornherein gesetzt ist, setzt das Größengadget in den rechten Rahmen.

sizeBBottom Wenn dieses Flag gesetzt ist, wird das Größengadget unterhalb des unteren Fensterrahmens angebracht. Das kann sinnvoll sein, wenn Sie bei der Textausgabe in der untersten Zeile nicht von dem Gadget gestört werden wollen.

simpleRefresh Ist dieses Flag gesetzt, so wird dem Programm über IDCMP mitgeteilt, wenn Ihr Window erneuert werden muß, z.B. wenn Teile davon von anderen Windows verdeckt wurden. Die Erneuerung müssen Sie allerdings selbst vornehmen, eine nicht zu unterschätzende Arbeit, die nur mit Speicherersparnis belohnt wird.

superBitmap Mit diesem Flag kann der Programmierer die Steuerung seiner Bitmap selbst, z.B. eine Bitmap über 1024∗1024 Bildpunkte ist nur über diese Methode zu realisieren. Das Feld BitMap der New-

Window-Struktur muß dann einen Zeiger auf die eigene Bitmap enthalten.

backDrop Geben Sie Ihrem Window dieses Flag mit auf den Weg, so sorgt Intuition dafür, daß es stets das zuunterst liegende Fenster ist. Als Systemgadget ist nur noch das Schließsymbol erlaubt. Besonders gut macht es sich in Malprogrammen wie DPaint, wo alle anderen Fenster (Symbolleiste, Farbpalette, ...) über dem eigentlichen Zeichenwindow geöffnet werden.

reportMouse Dieses Flag muß gesetzt werden, um die aktuelle Mausposition im Window-Record (s.u.) auslesen zu können. Ferner muß dieses Flag gesetzt sein, um mit dem IDCMP-Flag MOUSEMOVE arbeiten zu können.

gimmeZeroZero Durch Setzen dieses Flags werden Fensterrahmen und -inhalt getrennt. Der Punkt (0;0), der normalerweise in der linken, oberen Ecke des Fensterrahmens liegt, sitzt bei einem GZZ-Window in der linken, oberen Ecke der Zeichenebene; der Fensterrahmen samt Titelleiste bekommt einen getrennten Rastport, siehe dazu die Besprechung des Window-Records.

borderless Setzen Sie dieses Flag, schalten Sie den Fensterrahmen (nicht die Titelleiste!) ab. Ganz besonders in Kombination mit BACKDROP können Sie Windows für Grafikprogramme erzeugen, die den Benutzer nicht mit irgendwelchen Rahmen stören.

activate Das Fenster wird sofort nach dem Öffnen aktiviert.

windowActive Intuition setzt dieses Flag, wenn Ihr Window aktiv ist.

inRequest Intuition setzt dieses Flag, wenn Ihr Window im Request-Modus ist.

menuState Intuition setzt dieses Flag, wenn Ihr Fenster ein Menü auslöst.

rmbTrap Wenn Sie in Ihrem Programm keine Menüs haben und die rechte Maustaste für etwas anderes verwenden wollen (denn sie löst ja normalerweise die Menüauswahl aus), dann setzen Sie am besten dieses Flag. Ein Druck auf die rechte Maustaste liefert so eine MOUSEBUTTONS-Meldung aus, vorausgesetzt, daß IDCMP-Flag MOUSEBUTTONS ist gesetzt (s.u.)

noCareRefresh Ist dieses Flag gesetzt, erhalten Sie keine Meldung, wenn Ihr Window erneuert werden muß.

wf18..wf23	Für das System reserviert.
windowRefresh	Ihr Fenster wird gerade erneuert (wird von Intuition gesetzt).
wbenchWindow	Ihr Window ist das einzige auf der Workbench (wird von Intuition gesetzt).
windowTicked	Zur Zeit nur ein Timertick (wird von Intuition gesetzt).
wf27..wf31	Für das System reserviert
idcmpFlags	Hier kann man angeben, über welche Aktionen des Benutzers (Anklicken des Schließsymbols oder eines eigenen Gadgets, Menüauswahl, ...) der Programmierer Nachricht erhalten soll. Zulässig sind die Werte aus der Menge IDCMPFlags:
sizeVerify	Dieses Flag ist zu setzen, wenn das Programm in einem Window Grafiken zeichnet oder Texte schreibt, und dieses Zeichnen abgeschlossen werden muß, ehe der Benutzer das Window vergrößert oder verkleinert. Sollte er es versuchen, wartet Intuition, bis das betreffende Programm antwortet.
newSize	Das Programm erhält von Intuition Nachricht, nachdem der Benutzer das Fenster in seiner Größe verändert hat, im Gegensatz zu SIZEVERIFY, wo das Programm Nachricht erhält, wenn der Benutzer eine Größenänderung durchführt (die Fenstergröße muß sich aber nicht verändern).
refreshWindow	Das Programm erhält Nachricht, wenn ein Refresh (Erneuerung) notwending ist, d.h. wenn das Fenster z.B. durch ein anderes überlappt wird.
mouseButtons	Wenn dieses Flag gesetzt ist, erhält das Programm Meldung, ist eine Maustaste betätigt worden.
mouseMove	Sollte dieses Flag gesetzt sein, erhält das Programm Nachricht, wenn die Maus bewegt wurde. Dieses Flag wirkt nur, wenn Sie bei den Window-Flags REPORTMOUSE oder bei einem Gadget FOLLOWMOUSE gesetzt haben. Die neue Position kann in den Feldern MOUSEX und MOUSEY des Window-Records (s.u.) abgefragt werden.
gadgetDown	Das Programm erhält Meldung, wenn ein Gadget des Typs GADGIMMEDIATE betätigt wurde (keine Panik, Gadgets werden in diesem Buch auch noch besprochen).

gadgetUp Wenn der Benutzer ein Gadget des Typs RELVERIFY betätigt und dieses Flag gesetzt ist, erhält Ihr Programm darüber Meldung.

reqSet Setzen Sie dieses Flag, wenn Sie Nachricht erhalten wollen, wenn der erste Requester in diesem geöffnet wird.

menuPick Haben Sie dieses Flag gesetzt, erhalten Sie Nachricht darüber, wenn der Benutzer die rechte Maustaste (die Menütaste) betätigt hat. Wie Menü und Menüpunkt abgefragt werden, soll in einem eigenen Kapitel besprochen werden.

closeWindow Sollten Sie dieses Flag gesetzt und der Benutzer das Schließsymbol angeklickt haben, erhalten Sie darüber Nachricht und können entsprechend reagieren (Programmende, Schließen des Fensters, ...)

rawKey Das Programm erhält Nachricht darüber, wenn eine Taste gedrückt wurde. Dieses Flag ist wohl nur sinnvoll, wenn Tastatureingaben im Window zugelassen sind, und das Programm jede Taste selbst interpretieren soll.

reqVerify Setzen Sie dieses Flag, um sicherzugehen, daß alle Zeichenoperationen im Window abgeschlossen wurden, bevor ein Requester eröffnet wird.

reqClear Wenn Sie dieses Flag setzen, erhalten Sie Nachricht, wenn der letzte Requester geschlossen wurde.

menuVerify Ist dieses Flag gesetzt, so wartet Intuition darauf, daß Ihr Programm alle Zeichenoperationen im betreffenden Window abgeschlossen hat, bevor es erlaubt, Menüoperationen durchzuführen.

newPrefs Das Programm erhält Nachricht, wenn die Systemvoreinstellungen über das Preferences-Programm oder die Preferences-Struktur geändert wurden.

diskInserted Sie erhalten Meldung, wenn eine Diskette in ein beliebiges Laufwerk eingelegt wurde.

diskRemoved Sie erhalten Meldung, wenn eine Diskette aus einem beliebigen Laufwerk herausgenommen wurde.

wbenchMessage Wird intern verwendet.

activeWindow Ist dieses Flag gesetzt, erhält Ihr Programm Nachricht darüber, wenn das betreffende Window aktiviert (also angeklickt) wurde.

inactiveWindow Ist dieses Flag gesetzt, erhält Ihr Programm Nachricht darüber, wenn das betreffende Window deaktiviert wurde.

deltaMove Wenn dieses Flag gesetzt ist, wird in die Felder MOUSEX und MOUSEY des Window-Records (s.u.) statt der neuen absoluten Mausposition die Differenz zur alten angegeben. DELTAMOVE funktioniert nur in Verbindung mit MOUSEMOVE.

vanillaKey Haben Sie dieses Flag gesetzt, werden Tastatureingaben nicht als Amiga- sondern als ASCII-Tastaturcodes übergeben.

intuiTicks Ihr Programm bekommt alle 1/50stel Sekunde (IntuiTick) eine Nachricht.

c23..c30 Für das System reserviert.

lonelyMessage Wird intern verwendet.

Voraussetzung für das einwandfreie Funktionieren der Flags ist, wenn man von ACTIVEWINDOW und INACTIVEWINDOW einmal absieht, da das betreffende Fenster aktiviert sein muß (über Anklicken oder Setzen des ACTIVATE-Flags zu erreichen).

3.3.4 Die Window-Struktur

Nach dem Aufruf von OPENWINDOW (ob aus WINDOWS oder INTUITION ist egal) zeigt der Rückgabewert auf den Window-Record, der viele interessante Daten enthält:

```
Window=RECORD
  nextWindow   : WindowPtr      ; Zeiger auf das nächste Window
  leftEdge     : INTEGER        ; x-Koordinate der linken, oberen Ecke des Windows
  topEdge      : INTEGER        ; y-Koordinate der linken, oberen Ecks des Windows
  width        : INTEGER        ; Breite des Windows
  height       : INTEGER        ; Höhe des Windows
  mouseY       : INTEGER        ; akt. y-Koordinate der Maus
                                  (relativ zum Window!!);
                                  REPORTMOUSE muß gesetzt sein!
  mouseX       : INTEGER        ; akt. x-Koordinate der Maus (relativ zum
                                  Window!!);
                                  REPORTMOUSE muß gesetzt sein!
  minWidth     : INTEGER        ; minimale Breite des Windows
  minHeight    : INTEGER        ; minimale Höhe des Windows
  maxWidth     : INTEGER        ; maximale Breite des Windows
  maxHeight    : INTEGER        ; maximale Höhe des Windows
```

```
  flags       : WindowFlagSet  ; Flags des Windows
  menuStrip   : MenuPtr        ; Zeiger auf das erste Menü
  title       : ADDRESS        ; Zeiger auf den Titel-String
  firstRequest: RequesterPtr   ; Zeiger auf den ersten Requester
  dmRequest   : RequesterPtr   ; Zeiger auf den ersten Double-Click-Requester
  reqCount    : INTEGER        ; Anzahl der Requester
  wScreen     : ScreenPtr      ; Zeiger auf den Screen des Windows
  rPort       : RastPortPtr    ; Zeiger auf den Rastport des Windows (für Grafik
                                 notwendig)
  borderLeft  : Byte           ; linker Fensterrahmen
  borderTop   : Byte           ; oberer Fensterrahmen
  borderRight : Byte           ; rechter Fensterrahmen
  borderBottom: Byte           ; unterer Fensterrahmen
  borderRPort : RastPortPtr    ; Zeiger auf den Rastport des Window-Rahmens
  firstGadget : GadgetPtr      ; Zeiger auf das erste Gadget des Fensters
  parent      : WindowPtr      ; Zeiger auf das übergeordnete Fenster
  descendant  : WindowPtr      ; Zeiger auf das untergeordnete Fenster
  pointer     : ADDRESS        ; Zeiger auf die Daten des Mauspointers
  ptrHeight   : Byte           ; Höhe des Mauspointers
  ptrWidth    : [0..16]        ; Breite des Mauspointers
  xOffset     : Byte           ; x-Abstand des Pointers zum Nullpunkt
  yOffset     : Byte           ; y-Abstand des Pointers zum Nullpunkt
  idcmpFlags  : IDCMPFlagSet   ; gesetzte IDCMPs
  userPort    : MsgPortPtr     ; User-Port des Fensters, für IDCMP-Abfrage wichtig
  windowPort  : MsgPortPtr     ; Nachrichten-Port des Fensters
  messageKey  : IntuiMessagePtr; Zeiger auf den Nachrichten-Record des Fensters
  detailPen   : Byte           ; Detail-Pen des Fensters
  blockPen    : Byte           ; Block-Pen des Fensters
  checkMark   : ImagePtr       ; Zeiger auf den Haken, der bei manchen
                                 Menüfeldern erscheint, um anzuzeigen daß
                                 die entsprechende Option angewählt ist.
  screenTitle : ADDRESS        ; Zeiger auf den Titel des Screens
  gzzMouseX   : INTEGER        ; akt. x-Koordinate der Maus (relativ zum
                                 Fensterinhalt, nicht aber zum Rahmen!!)
  gzzMouseY   : INTEGER        ; akt. y-Koordinate der Maus (relativ zum
                                 Fensterinhalt, nicht aber zum Rahmen!!)
  gzzWidth    : INTEGER        ; Breite des Fensterinhaltes (der Zeichenebene
                                 also), nicht aber die des ganzen Fensters!
  gzzHeight   : INTEGER        ; Höhe der Zeichenebene, nicht aber die des
                                 gesamten Fensters!
  extData     : ADDRESS        ; Für spätere Betriebssystemversion.
  userData    : ADDRESS        ; Für zur Verwendung des Benutzers.
  wLayer      : LayerPtr       ; Zeiger auf die Layer-Struktur des Windows
  iFont       : TextFontPtr    ; Zeiger auf die Zeichensatz-Struktur des
                                 Zeichensatzes im Fenster
END;
```

Hier sind aber einige Felder noch unklar. Doch keine Angst, sie sollen erläutert werden; die Informationsvielfalt, die in diesem Record steckt, sollte man sich wirklich nicht entgehen lassen:

borderLeft, borderTop, borderRight, borderBottom — In diesen Feldern stehen die Dimensionen der Rahmen in Bildpunkten. Der obere Rahmen hat normalerweise den Wert 11, der linke 4.

borderRPort — Bei GIMMEZEROZERO-Fenstern ist das der RastPort für den Fensterrahmen, andernfalls NIL. Was ein RastPort überhaupt ist, erfahren Sie im GRAPHICS-Kapitel.

parent — Dieser Zeiger zeigt auf das Vorgängerfenster der Liste. Ist das Fenster das erste, ist dieses Feld NIL.

descendant — Dieses Feld zeigt auf das Nachfolgerfenster der internen Liste. Beim letzten Fenster ist dieses Feld NIL.

gzzMouseX, gzzMouseY, gzzWidth, gzzHeight — Diese Felder sind nur bei GIMMEZEROZERO-Fenstern interessant und geben die Mauskoordinaten relativ zur Zeichenebene und deren Ausmaße an.

Zum Öffnen von Windows nach dieser Art noch ein Beispielprogramm, das zwei Windows mit verschiedenen Flags öffnet:

```
MODULE WindowDemo2;

FROM SYSTEM     IMPORT ADR;
FROM Intuition  IMPORT NewScreen,ScreenPtr,OpenScreen,CloseScreen,customScreen,
                       OpenWindow,CloseWindow,NewWindow,WindowPtr,WindowFlags,
                       WindowFlagSet,IDCMPFlagSet;
FROM Graphics   IMPORT ViewModes,ViewModeSet;
FROM Dos        IMPORT Delay;

VAR MeinScreen                : ScreenPtr;
    MeinWindow1,MeinWindow2   : WindowPtr;
    ScreenDaten               : NewScreen;
    W1Daten,W2Daten           : NewWindow;

BEGIN
  WITH ScreenDaten DO (* Screendaten initialisieren *)
    leftEdge:=0; topEdge:=0; width:=640; height:=256;
    depth:=2; detailPen:=0; blockPen:=1; viewModes:=ViewModeSet{hires};
    type:=customScreen; font:=NIL; defaultTitle:=ADR("TEST-SCREEN");
```

```
    gadgets:=NIL; customBitMap:=NIL;
  END;

  WITH W1Daten DO (* Daten für's erste Window *)
    leftEdge:=5Ø; topEdge:=5Ø; width:=25Ø; height:=8Ø;
    detailPen:=Ø; blockPen:=1; idcmpFlags:=IDCMPFlagSet{};
    flags:=WindowFlagSet{noCareRefresh,windowDrag}; firstGadget:=NIL;
    checkMark:=NIL; title:=ADR("noCareRefresh"); bitMap:=NIL;
    minWidth:=Ø; maxWidth:=Ø; minHeight:=Ø; maxHeight:=Ø;
    type:=customScreen;
  END;

  WITH W2Daten DO (* Daten für's zweite Window *)
    leftEdge:=5Ø;topEdge:=14Ø; width:=25Ø; height:=8Ø;
    detailPen:=Ø; blockPen:=1; idcmpFlags:=IDCMPFlagSet{};
    flags:=WindowFlagSet{windowDrag,windowDepth,windowSizing,activate,
    windowRefresh};
    firstGadget:=NIL; checkMark:=NIL; title:=ADR("windowRefresh u.a.");
    bitMap:=NIL; minWidth:=Ø; maxWidth:=Ø; minHeight:=Ø; maxHeight:=Ø;
    type:=customScreen;
  END;

  MeinScreen:=OpenScreen(ScreenDaten); (* Öffnen *)

  IF MeinScreen#NIL THEN
    W1Daten.screen:=MeinScreen;
    W2Daten.screen:=MeinScreen;
    MeinWindow1:=OpenWindow(W1Daten);
    IF MeinWindow1#NIL THEN
      Delay(25Ø);
      MeinWindow2:=OpenWindow(W2Daten);
      IF MeinWindow2#NIL THEN
        Delay(5ØØ);

        (* An dieser Stelle könnten Sie eigene Programmteile einfügen *)

        CloseWindow(MeinWindow2);
      END;
      CloseWindow(MeinWindow1);
    END;
    CloseScreen(MeinScreen);
  END
END WindowDemo2.
```

3.3.5 Window-Befehle

Analog zum OPENWINDOW- gibt es auch den CLOSEWINDOW-Befehl:

CloseWindow (MeinWindow);

MeinWindow	WindowPtr auf das zu schließende Fenster.

Jedes Fenster sollte spätestens am Programmende geschlossen werden, damit der Speicherplatz, den es belegt, auch wieder freigegeben wird.

Um ein Fenster nachträglich zu aktivieren, gibt es den Befehl ACTIVATEWINDOW:

ActivateWindow (MeinWindow);

MeinWindow	WindowPtr auf das zu aktivierende Fenster.

Sollte man über IDCMP die Nachricht bekommen, daß ein Window erneuert werden muß (s.o.), so kann er es mit BEGINREFRESH erneuern:

BeginRefresh (MeinWindow);

MeinWindow	WindowPtr auf das Window, welches erneuert werden soll. Während ein Fenster erneuert wird, setzt Intuition das Flag WINDOWREFRESH und ignoriert sämtliche Befehle, die sich nicht auf Bereiche innerhalb des Windows beziehen. BEGINREFRESH ist sicher nur für SIMPLEREFRESH-Fenster sinnvoll, da sonst (also bei keinem gesetzten Refresh-Flag) Intuition die Erneuerung selbst übernimmt.

Damit Intuition wieder in den Normalzustand versetzt wird, ist der Befehl ENDREFRESH zu verwenden:

EndRefresh (MeinWindow , complete);

MeinWindow	WindowPtr auf das Window, welches erneuert worden ist.
complete	BOOLEAN-Wert, der angibt, ob das Fenster vollständig erneuert werden konnte, TRUE steht für ja und FALSE für nein.

Möchte man aber nicht das ganze Fenster, sondern nur den Rahmen erneuern, benutze man den Befehl REFRESHWINDOWFRAME:

RefreshWindowFrame (MeinWindow);

MeinWindow	WindowPtr auf das Fenster, dessen Rahmen erneuert werden soll.

Um die IDCMP-Flags eines Windows zu ändern, z.B. wenn man nachträglich ein Menü einbauen möchte, gibt es den Befehl MODIFYIDCMP:

ModifyIDCMP (MeinWindow , MeineIDCMP);

MeinWindow	WindowPtr auf das Window, dessen IDCMPs geändert werden sollen.
MeineIDCMP	IDCMPFlagSet-Wert, der die neuen IDCMP-Flags enthält.

Wie man Screens bewegen kann, so ist dies auch mit Windows möglich, allerdings nicht nur rauf und runter, sondern auch quer über den Bildschirm können die Fenster bewegt werden:

MoveWindow (MeinWindow , deltax , deltay);

MeinWindow	WindowPtr auf das Window, welches verschoben werden soll.
deltax	INTEGER-Wert, um wieviel Bildpunkte das Fenster in x-Richtung bewegt werden soll.
deltay	INTEGER-Wert, um wieviel Bildpunkte das Fenster in y-Richtung bewegt werden soll.

Sind »deltax« und/oder »deltay« zu groß, so wird das Fenster um die größtmögliche Anzahl von Bildpunkten verschoben.

Um die Priorität eines Windows zu verändern, gibt es die Tiefen-Gadgets rechts oben. Es gibt aber auch äquivalente Befehle, WINDOWTOFRONT, um ein Fenster nach vorn zu holen und WINDOWTOBACK für das Gegenteil:

WindowToFront (MeinWindow);

MeinWindow	WindowPtr auf das Fenster, welches nach vorn geholt werden soll.

WindowToBack (MeinWindow);

MeinWindow	WindowPtr auf das Fenster, welches nach hinten geschoben werden soll.

Doch nicht nur die Priorität, auch die Größe eines Fensters läßt sich im Nachhinein ändern:

SizeWindow (MeinWindow , deltax , deltay);

MeinWindow	WindowPtr auf das Window, dessen Größe geändert werden soll.
deltax	INTEGER-Wert der Bildpunkte, um die das Fenster in x-Richtung vergrößert werden soll.
deltay	INTEGER-Wert der Bildpunkte, um die das Fenster in y-Richtung vergrößert werden soll.

Möchte man die Maximal- und Minimalgröße eines Windows nach dem Öffnen ändern, bietet sich der WINDOWLIMITS-Befehl an:

ok := WindowLimits (MeinWindow , minx , miny , maxx , maxy);

ok	BOOLEAN-Rückgabewert, ob die Größenbeschränkung erfolgreich verlaufen ist; TRUE für ja und FALSE für nein.
MeinWindow	WindowPtr auf das Window, dessen Minimal- und Maximalgrößen geändert werden sollen.
minx	neue Minimalgröße in x-Richtung
miny	neue Minimalgröße in y-Richtung
maxx	neue Maximalgröße in x-Richtung
maxy	neue Maximalgröße in y-Richtung

Eine gern verwendete Möglichkeit, den Benutzer auf etwas aufmerksam zu machen, ist die Anzeige eines Textes in der Titelleiste des Fensters. Doch in der NewWindow-Struktur haben wir eigentlich immer den Programm- oder Windownamen, nicht aber irgendeine Meldung stehen. Es muß also eine Möglichkeit geben, den Titeltext des Fensters zu verändern und die heißt SETWINDOWTITLES:

SetWindowTitles (MeinWindow , Windowtitel , Screentitel);

MeinWindow	WindowPtr auf das Window, dessen Titeltext geändert werden soll.
Windowtitel	ADDRESS-Wert des Strings mit dem neuen Fenstertitel.
Screentitel	ADDRESS-Wert des Strings mit dem neuen Titel des Screens des Fensters.

Für »Windowtitel« und »Screentitel« können aber auch die Werte 0 und –1 angegeben werden. Diese bedeuten:

0:	Der entsprechende Titel wird gelöscht.
–1:	der entsprechende Titel bleibt erhalten.

Will man die Titelleiste eines Screens in den Hintergrund von BACKDROP-Windows verbannen, genügt auch hier ein Befehl:

ShowTitle (MeinScreen , showIt);

MeinScreen	ScreenPtr auf den Screen, dessen Titelleiste ausgeblendet werden soll.
showIt	BOOLEAN-Wert der angibt, ob die Titelleiste sichtbar (TRUE) oder nicht sichtbar (FALSE) ist.

Wenn man nach dem Öffnen von Fenstern noch Informationen über die Mausbewegungen erhalten will, das REPORTMOUSE-Flag in der NewWindow-Struktur aber nicht gesetzt hat, besteht die Möglichkeit, einen entsprechenden Befehl zu verwenden, nämlich REPORTMOUSE (wie der Name des Flags):

ReportMouse (SetOrNot , MeinWindow);

SetOrNot	BOOLEAN-Wert, enthält er TRUE, so wird REPORTMOUSE eingeschaltet, bei FALSE wird sie ausgeschaltet.
MeinWindow	WindowPtr auf das Fenster, dem die Mausinformation übergeben werden soll. (Siehe Erklärung der Window-Flags, REPORTMOUSE.)

Haben Sie in einigen Programmen schon mal andere Mauszeiger als den, den Sie in den Preferences definiert haben, gesehen, bei DPAINT, CARRIER COMMAND oder

anderswo? Im Grunde ist so eine Veränderung recht einfach, muß man doch nur den Befehl SETPOINTER verwenden:

SetPointer (MeinWindow , PointerDaten , Höhe , Breite , x , y);

MeinWindow	WindowPtr auf das Window, dessen Pointer modifiziert werden soll.
PointerDaten	ADDRESS-Wert auf die Daten des neuen Pointers.
Höhe	INTEGER-Wert, der die Höhe des neuen Zeigers in Bildpunkten enthält.
Breite	INTEGER-Wert, der die Breite des neuen Zeigers in Bild-Punkten enthält.
x	INTEGER-Wert, der die x-Koordinate des HotSpots relativ zur linken, oberen Ecke des Pointers enthält. Der HotSpot ist der für Intuition einzig relevante Punkt des ganzen Mauszeigers, es kann nur etwas angeklickt werden, was sich unter dem HotSpot befindet.
y	INTEGER-Wert, der die y-Koordinate des HotSpots relativ zur linken, oberen Ecke des Zeigers enthält.

Um einen selbstdefinierten Pointer wieder zu löschen, benutzt man am besten CLEAR-POINTER:

ClearPointer (MeinWindow);

MeinWindow	WindowPtr auf das Fenster, dessen Mauszeiger gelöscht werden soll.

3.3.6 IDCMP-Abfrage

Auf dieses Unterkapitel wurde schon mehrmals verwiesen, nun ist es so weit. Bevor wir jedoch mit der IDCMP-Abfrage beginnen können, müssen wir uns klar machen, was IDCMP überhaupt ist.

IDCMP (Intuition Direct Communication Message Ports – Intuition's direkte Kommunikations-Nachrichten-Ports) muß man sich wie einen Filter vorstellen. Mit den IDCMP-Flags des NewWindow-Record läßt sich bestimmen, welche Ereignisse (Anklicken des Schließsymbols, Drücken eines Mausknopfes, ...) den Programmierer erreichen sollen. Alle anderen werden »durchgelassen» und der Programmierer erfährt nichts von ihnen.

Das ganze Nachrichtensystem läuft nun über den UserPort des Fensters, der für die Nachrichten zuständig ist. Mit

WaitPort (MeinUserPort);

MeinUserPort — MsgPortPtr auf einen Nachrichtenport, hier am besten MeinWindow^.userPort

setzen wir unser Programm auf eine Warteliste. Es wird nun solange nicht fortgeführt, bis im angegebenen Nachrichtenport eine Nachricht eingetroffen ist. Diese fragen wir mit GETMSG ab:

MeineMessage := GetMsg (MeinUserPort);

MeinUserPort — UserPort, aus dem die Nachricht gelesen werden soll.

MeineMessage — IntuiMessagePtr auf den Record, in den die Nachricht eingelesen werden soll:

```
IntuiMessage=RECORD
  execMessage: Message        ; entsprechende EXEC-Message (siehe EXEC-Kapitel)
  class      : IDCMPFlagSet   ; Art der Nachricht (bedingt durch gewählte IDCMPs)
  code       : CARDINAL       ; Feld für Spezialinformationen, z.B. eine
                                Menünummer
  qualifier  : CARDINAL       ; Qualifier der Nachricht, eine weitere
                                Spezialinformation
  iAddress   : ADDRESS        ; Spezialfeld für evt. Adressen (z.B. die eines
                                angeklickten Gadgets)
  mouseX     : INTEGER        ; x-Koordinate der akt. Mausposition, für
                                MOUSEMOVE-IDCMP (relativ zur linken, oberen
                                Fensterecke)
  mouseY     : INTEGER        ; y-Koordinate der akt. Mausposition, für
                                MOUSEMOVE-IDCMP (relativ zur linken, oberen
                                Fensterecke)
  seconds    : LONGCARD       ; Systemsekunden (siehe DOS-Kapitel)
  micros     : LONGCARD       ; Systemmikrosekunden (siehe DOS-Kapitel)
  idcmpWindow: WindowPtr      ; Zeiger auf das Fenster, in dem das Ereignis, das
                                eine Nachricht zur Folge hatte, passiert ist
  specialLink: IntuiMessagePtr; wird vom System benutzt
END;
```

Das im Moment einzig für uns wichtige Feld ist CLASS. Es enthält das IDCMP-Flag des Ereignisses (oder die Flags der Ereignisse), das (die) eine Message auslöste(n), z.B. CLOSEWINDOW beim Anklicken des Schließsymbols, MOUSEBUTTONS beim

Maustastendruck. Es empfiehlt sich, dieses Feld in einer Variable zwischenzuspeichern, da es beim nächsten Befehl gelöscht wird, beim Beantworten einer Nachricht. Um Intuition (oder EXEC) zu zeigen, daß wir die Message zu Kenntnis genommen haben, müssen wir sie beantworten, und zwar mit REPLYMSG:

ReplyMsg (MeineIntuiMessage);

MeineIntui	MessageIntuiMessagePtr auf die zu beantwortende Nachricht.

Dann müssen wir nur noch die Ereignisse auswerten und unser Programm entsprechend reagieren lassen, z.B.

```
IF (closeWindow IN class) THEN ...
(Schließsymbol?)
```

oder

```
IF (mouseButtons IN class) THEN ...
(Maustaste?)
```

Damit's anschaulicher ist, soll nun ein Beispielprogramm folgen:

```
MODULE IDCMPDemo;

FROM SYSTEM     IMPORT ADR;
FROM Intuition  IMPORT NewScreen,ScreenPtr,OpenScreen,CloseScreen,customScreen,
                       OpenWindow,CloseWindow,NewWindow,WindowPtr,WindowFlags,
                       WindowFlagSet,IDCMPFlagSet,IDCMPFlags,IntuiMessage;
FROM Graphics   IMPORT ViewModes,ViewModeSet;

FROM Dos        IMPORT Delay;
FROM Exec       IMPORT GetMsg,ReplyMsg,WaitPort;

VAR MeinScreen              : ScreenPtr;
    MeinWindow1,MeinWindow2 : WindowPtr;
    ScreenDaten             : NewScreen;
    W1Daten,W2Daten         : NewWindow;
    class                   : IDCMPFlagSet;
    IntuiMsg                : POINTER TO IntuiMessage;

BEGIN
  WITH ScreenDaten DO (* Screendaten initialisieren *)
    leftEdge:=0; topEdge:=0; width:=640; height:=256;
    depth:=2; detailPen:=0; blockPen:=1; viewModes:=ViewModeSet{hires};
    type:=customScreen; font:=NIL; defaultTitle:=ADR("TEST-SCREEN");
    gadgets:=NIL; customBitMap:=NIL;
  END;
```

```
  WITH W1Daten DO (* Daten für's erste Window *)
    leftEdge:=50; topEdge:=50; width:=250; height:=80;
    detailPen:=0; blockPen:=1; idcmpFlags:=IDCMPFlagSet{};
    flags:=WindowFlagSet{noCareRefresh,windowDrag}; firstGadget:=NIL;
    checkMark:=NIL; title:=ADR("kein Closegadget"); bitMap:=NIL;
    minWidth:=0; maxWidth:=0; minHeight:=0; maxHeight:=0;
    type:=customScreen;
  END;

  WITH W2Daten DO (* Daten für's zweite Window *)
    leftEdge:=50;topEdge:=140; width:=250; height:=80;
    detailPen:=0; blockPen:=1; idcmpFlags:=IDCMPFlagSet{closeWindow};
    flags:=WindowFlagSet{windowDrag,windowDepth,windowSizing,activate,
    windowRefresh,windowClose};
    firstGadget:=NIL; checkMark:=NIL; title:=ADR("<-- Closegadget");
    bitMap:=NIL; minWidth:=0; maxWidth:=0; minHeight:=0; maxHeight:=0;
    type:=customScreen;
  END;

  MeinScreen:=OpenScreen(ScreenDaten); (* Öffnen *)

  IF MeinScreen#NIL THEN
    W1Daten.screen:=MeinScreen;
    W2Daten.screen:=MeinScreen;
    MeinWindow1:=OpenWindow(W1Daten);
    IF MeinWindow1#NIL THEN
      MeinWindow2:=OpenWindow(W2Daten);
      IF MeinWindow2#NIL THEN
        LOOP
          WaitPort(MeinWindow2^.userPort); (* Warten bis Nachricht eintrifft *)
          IntuiMsg:=GetMsg(MeinWindow2^.userPort); (* Nachricht holen *)
          WHILE IntuiMsg#NIL DO (* Auswerten *)
            class:=IntuiMsg^.class; (* Art der Nachricht *)
            ReplyMsg(IntuiMsg); (* Antworten *)
            IF (closeWindow IN class) THEN EXIT; END; (* Ende, wenn
                                  Closegadget *)
            IntuiMsg:=GetMsg(MeinWindow2^.userPort);
          END;
        END;
        CloseWindow(MeinWindow2);
      END;
      CloseWindow(MeinWindow1);
    END;
    CloseScreen(MeinScreen);
  END
END IDCMPDemo.
```

Wenn Sie sich das Programm so ansehen, fällt Ihnen vielleicht auf, daß die Abfrageroutine für mehrere gleichzeitig gesetzte IDCMPs geeignet ist, doch ist denn für ein einziges, lausiges IDCMP-Flag so ein Aufwand nötig? Sicher nicht. Wir können auch abfragen, ob überhaupt eine Message eingetroffen ist und bei einem einzigen gesetzen IDCMP ist das Spektrum ja nicht sehr groß. Sehen Sie sich doch einmal folgendes Programm an:

```
MODULE IDCMPDemo2;

FROM SYSTEM    IMPORT ADR,LONGSET;
FROM Intuition IMPORT NewScreen,ScreenPtr,OpenScreen,CloseScreen,customScreen,
                      OpenWindow,CloseWindow,NewWindow,WindowPtr,WindowFlags,
                      WindowFlagSet,IDCMPFlagSet,IDCMPFlags,IntuiMessage;
FROM Graphics  IMPORT ViewModes,ViewModeSet;
FROM Exec      IMPORT SetSignal;

VAR MeinScreen : ScreenPtr;
    MeinWindow : WindowPtr;
    ScreenDaten: NewScreen;
    WDaten     : NewWindow;

BEGIN
  WITH ScreenDaten DO (* Screendaten initialisieren *)
    leftEdge:=0; topEdge:=0; width:=640; height:=256;
    depth:=2; detailPen:=0; blockPen:=1; viewModes:=ViewModeSet{hires};
    type:=customScreen; font:=NIL; defaultTitle:=ADR("TEST-SCREEN");
    gadgets:=NIL; customBitMap:=NIL;
  END;

  WITH WDaten DO (* Daten für's Window *)
    leftEdge:=50;topEdge:=50; width:=539; height:=80;
    detailPen:=0; blockPen:=1; idcmpFlags:=IDCMPFlagSet{mouseButtons};
    flags:=WindowFlagSet{windowDrag,windowDepth,windowSizing,activate, windowRe-
fresh,windowClose};
    firstGadget:=NIL; checkMark:=NIL; title:=ADR("Mausknopf drücken!");
    bitMap:=NIL; minWidth:=0; maxWidth:=0; minHeight:=0; maxHeight:=0;
    type:=customScreen;
  END;

  MeinScreen:=OpenScreen(ScreenDaten); (* Öffnen *)

  IF MeinScreen#NIL THEN
    WDaten.screen:=MeinScreen;
    MeinWindow:=OpenWindow(WDaten);
    IF MeinWindow#NIL THEN
      WHILE NOT (MeinWindow^.userPort^.sigBit IN SetSignal(LONGSET{}, LONGSET{})) DO
      END; (* Warteschleife, bis eine Message eingetroffen ist *)
```

```
      CloseWindow(MeinWindow);
    END;
    CloseScreen(MeinScreen);
  END
END IDCMPDemo2.
```

Zwischen dem WHILE NOT und dem dazugehörigen END könnte irgendein tolles Programm bis zum Maustastendruck laufen (siehe Demoprogramm »SPARKS« auf der M2Amiga-Diskette). Bei dieser Art von IDCMP-Abfrage läuft der Programmteil zwischen WHILE und END solange durch, bis irgendeine Message eingetroffen ist. Im Beispielprogramm kann es ja nur ein Maustastendruck sein. Es würde den Rahmen dieses Kapitels sprengen, zu erklären, was es mit SIGBITs und Signalen auf sich hat. Bei Interesse schauen Sie bitte einmal im EXEC-Kapitel nach.

3.4 Gadgets

Sicher kennen Sie Gadgets (auf gut Deutsch Dingens, Apparat). Sie tragen einen großen Teil zur komfortablen Bedienung eines Programms bei, ist doch der Voreinsteller (Preferences) voll davon, z.B. bei der Einstellung der Farben bedienen Sie sich dreier Gadgets (in diesem Fall Proportionalgadgets oder Schieberegler). Auch bei den Fenstern gibt es Gadgets, die vier Systemgadgets nämlich: Schließsymbol, Größengadget, Tiefengadget und Ziehleiste (das Ding, mit dem Sie das Window bewegen). In diesem Kapitel soll erklärt werden, wie man eigene Gadgets programmiert.

3.4.1 Die Gadgettypen

Wir unterscheiden prinzipiell zwischen drei verschiedenen Gadgettypen:

1. Das Boolean-Gadget (kennt nur zwei verschiedene Arten: Angeklickt und nicht angeklickt, wird häufig als »OK»- oder »JA/NEIN»-Schalter benutzt).
2. Das Proportional-Gadget (Schieberegler, zur komfortablen Eingabe von Zahlen gedacht).
3. Das String-Gadget (zur komfortablen Eingabe von kleineren Texten wie Dateinamen o.ä. gedacht).

Als vierten Punkt könnte man noch das Integer-Gadget nennen, das aber nur eine Unterart des String-Gadgets ist; es läßt nämlich ausschließlich LONGINT-Werte zu. In dieser Reihenfolge wollen wir die Gadgets besprechen.

3.4.2 Der Gadget-Record

Der Gadget-Record ist eine Struktur, die alle drei Gadgettypen gemeinsam haben:

```
Gadget=RECORD
  nextGadget   : GadgetPtr        ; Zeiger auf das nächste Gadget
  leftEdge     : INTEGER          ; x-Koordinate der linken, oberen Gadgetecke
                                    (relativ zum Fenster!)
  topEdge      : INTEGER          ; y-Koordinate der linken, oberen Gadgetecke
                                    (relativ zum Fenster!)
  width        : INTEGER          ; Breite des Gadgets
  height       : INTEGER          ; Höhe des Gadgets
  flags        : GadgetFlagSet    ; Gadgetflags
  activation   : ActivationFlagSet; Aktivierungsflags
  gadgetType   : CARDINAL         ; Typ des Gadgets
  gadgetRender : ADDRESS          ; Zeiger auf den Gadgetrahmen oder ein Bild
  selectRender : ADDRESS          ; Zeiger auf neuen Rahmen (oder Bild)
  gadgetText   : IntuiTextPtr     ; Zeiger auf den Gadgettext
  mutualExclude: LONGSET          ; Maske für sich ausschließende Gadgets
  specialInfo  : ADDRESS          ; für String- und Proportionalgadgets
  gadgetID     : INTEGER          ; Nummer des Gadgets (kann vom Programmierer
                                    festgelegt werden)
  userData     : ADDRESS          ; Adresse einer eigenen Routine
END;
```

Da fast alle Felder im Kommentar nur unzureichend erklärt werden konnten folgt nun eine vollständige Erläuterung:

flags Dieses Feld enthält Angaben zur Art und Weise der Darstellung des Gadgets. Alle zulässigen Werte befinden sich in der Menge GadgetFlagSet:

gadgHBox Um den Bereich, den der Benutzer anklicken kann, wird ein Rahmen gezeichnet.

gadgHComp Der Klickbereich wird invertiert dargestellt (dieses Flag ist in GadgetFlagSet nicht enthalten, es wird von Intuition gesetzt, wenn die GadgetFlagSet-Menge leer ist).

gadgImage Es wird anstelle eines Textes ein Bild angezeigt, auf dessen Struktur (Image) gadgetRender zeigen muß. Paradebeispiel hierfür ist die Auswahlleiste von DPaint. Diese Option gilt auch für selectRender!

gadgHImage Es wird beim Anwählen des Gadgets das Objekt (Border oder Image) dargestellt, worauf selectRender zeigt.

gRelBottom — Ist dieses Flag gesetzt, gibt topEdge nicht die Koordinate relativ zum oberen sondern zum unteren Fensterrand an.

gRelRight — Setzen Sie dieses Flag, gibt leftEdge nicht die Koordinate relativ zum linken sondern zum rechten Fensterrand an.

gRelWidth — Mit dem Setzen dieses Flags erreichen Sie, daß sich width nicht auf die absolute Gadgetbreite sondern auf die Breite des Fensters weniger des Wertes in width bezieht.

gRelHeight — Wie GRELWIDTH, nur daß gRelHeight die Gadgethöhe beeinflußt.

selected — Ein TOGGLESELECT-Gadget (ein Schalter, der entweder auf »EIN« oder »AUS« steht), wird von Anfang an auf »EIN« gesetzt.

gadgDisabled — Wünschen Sie, daß das betreffende Gadget erstmal nicht anwählbar sein soll (aus welchen Gründen auch immer), so setzen Sie doch dieses Flag. Es wird dann »ghosted«, also grob punktiert dargestellt.

activation — Die Activation-Flags beeinflussen das Verhalten der Gadgets beim Anklicken. Es sind die Werte aus ActivationFlagSet erlaubt:

relVerify — Setzen Sie dieses Flag, so wird nach ~~dem Anklicken und (!)~~ Loslassen des Gadgets eine IDCMP-Meldung losgeschickt (das IDCMP-Flag GADGETUP muß gesetzt sein).

gadgImmediate — Ist dieses Flag gesetzt, so wird sofort beim Anklicken eine IDCMP abgeschickt (Flag GADGETDOWN muß gesetzt sein).

endGadget — Dieses Flag ist nur für Requester-Gadgets zuständig (zu den Requestern kommen wir noch). Der Benutzer muß, um den Requester wieder verschwinden zu lassen, ein Gadget mit diesem Flag anklicken.

followMouse — Wenn der Benutzer ein Gadget mit diesem Flag anwählt, so wird das Programm, immer wenn der Benutzer fortan die Maus bewegt, deren Position erfahren.

rightBorder — Bei diesem Flag wird das Gadget im rechten Fensterrahmen untergebracht (bei GIMMEZEROZERO-Fenstern).

leftBorder — Bei gesetztem LEFTBORDER-Flag wird das entsprechende Gadget im linken Fensterrahmen untergebracht (bei GIMMEZEROZERO-Fenstern).

bottomBorder — Setzen Sie dieses Flag, so wird das Gadget im unteren Fensterrahmen installiert (bei GIMMEZEROZERO-Fenstern).

topBorder — Das Gadget wird im oberen Fensterrahmen angebracht (bei GIMMEZEROZERO-Fenstern).

toggleSelect — Dieses Flag ist nur für Boolean-Gadgets. Beim Anklicken verändern sie durch dieses Flag seinen Zustand (entweder ein neuer Text, ein neues Bild, ...). Den Status des Gadgets (angewählt oder nicht) bestimmen Sie mit dem SELECTED-Flag (s.o.), gesetztes SELECTED bedeuted in der Regel angewählt.

stringCenter — Dieses Flag, das nur für String-Gadgets gültig ist, gibt an, daß ein eingegebener Text automatisch zentriert, d.h. in die Gadgetmitte gerückt wird.

stringRight — Dieses Flag (nur für String-Gadgets) bewirkt, daß ein eingegebener Text rechtsbündig, d.h. im rechten Teil des Gadgets erscheint.

longint — Damit definieren Sie ein String-Gadget als Integer-Gadget (nur LONGINT-Werte sind bei der Eingabe zulässig). Der Eingabespeicher des Integer-Gadgets (s.u.) muß bereits einen LONGINT-Wert beinhalten.

altKeymap — Für das betreffende String-Gadget wird eine andere Tastaturtabelle (beispielsweise amerikanisch im Fremdwortfeld eines Vokabeltrainers) definiert. In das StringInfo-Feld altKeyMap (s.u.) muß ein Zeiger auf die neue Tastaturtabelle stehen.

goolExtend — Vielleicht für zukünftige Versionen.

gadgetType — Typ des Gadgets; folgende, in INTUITION-Include als Konstanten definierte Werte sind zulässig:

boolGadget — Boolean-Gadget (Schalter; z.B. die »RETRY«- und »CANCEL«-Felder in den Systemrequester (»Please insert Disk ... in any Drive«).

Gadget0002 — Möglicherweise für kommende Versionen.

propGadget — Proportional-Gadget (Schieberegler; z.B die Gadgets für die Einstellung der Farben im Preferences-Programm).

strGadget — String-Gadget (Eingabefeld; z.B. das Feld für die Eingabe des Kommentars beim Menüpunkt »INFO« es »WORKBENCH«-Menüs).

sizing	Größengadget am rechten, unteren Fensterrand (nur für Intuition).
wDragging	Gadget zum Verschieben in der Titelleiste eines Fensters (nur für Intuition).
sDragging	(Unsichtbares) Gadget zum Rauf- und Runterziehen eines Screens (nur für Intuition).
wUpFront	Gadget, um ein Fenster nach vorn zu bringen (Tiefengadget; nur »für« Intuition).
sUpFront	Tiefengadget, um Screens nach vorn zu bringen (nur für Intuition).
wDownBack	Gadget, um ein Fenster nach hinten zu schieben (Tiefengadget; nur für Intuition).
sDownBack	Tiefengadget, um Screens nach hinten zu bringen (nur für Intuition).
close	Schließsymbol für Fenster (nur für Intuition).
reqGadget	Requester-Gadget.
gzzGadget	Gadget im Rahmen eines GIMMEZEROZERO-Fensters.
scrGadget	Gadget in einem Screen.
sysGadget	initialisiert die Systemgadgets der Gadgetliste.
gadgetRender	Wollen Sie einen Rahmen um Ihr Gadget haben (was meistens der Fall ist) oder ein Bild statt oder zu einem Text darstellen, so muß dieses Feld auf den Border- (für Rahmen) oder Image-Record (für Bild) zeigen (beides wird noch erklärt). Möchten Sie einen Rahmen verwenden, so müssen Sie Intuition das mitteilen, indem Sie das Flag GADGIMAGE im Feld flags setzen.
selectRender	Soll das Gadget beim Anklicken sein Aussehen verändern, so muß hier die Image- oder Border-Struktur für das neue Bild bzw. den neuen Rahmen enthalten sein. Vorraussetzung für das Funktionieren einer Änderung des Aussehens beim Anklicken ist das gesetzte GADGHIMAGE-Flag im Feld flags. selectRender muß gleichen Typs wie gadgetRender sein (entweder beides Border oder beides Image)!
gadgetText	IntuiTextPtr auf einen Record namens IntuiText, der uns noch häufig über den Weg laufen wird:

```
IntuiText=RECORD
  frontPen : UByte        ; Vordergrundfarbe
  backPen  : UByte        ; Hintergrundfarbe
  drawMode : DrawModeSet ; Zeichenmodus (s.u.)
  leftEdge : INTEGER      ; x-Koordinate der linken, oberen Ecke der Position,
                            wo die Textausgabe starten soll (relativ zum Objekt,
                            in dem der Text erscheinen soll)
  topEdge  : INTEGER      ; y-Koordinate der linken, oberen Ecke der Position,
                            wo die Textausgabe starten soll (relativ zum Objekt,
                            in dem der Text erscheinen soll)
  iTextFont: TextAttrPtr ; Zeiger auf die Textattribut-Struktur (siehe
                            DiskFont-Kapitel, momentan bitte NIL)
  iText    : ADDRESS      ; Zeiger auf den String mit dem eigentlichen Text
  nextText : IntuiTextPtr; Zeiger auf den nächsten IntuiText
END;
```

Wahrscheinlich ist das Feld drawMode noch unklar:

drawMode — Hier wird der Zeichenmodus des Textes bestimmt. Zulässig sind die beiden in GRAPHICS definierten Konstanten JAM1 und JAM2 sowie die Werte der Menge DrawModeSet:

jam1 — Der Text wird in der mit frontPen definierten Farbe auf den Bildschirm gebracht. Die Punkte, die sich schon an der Stelle, wo der Text erscheinen soll, befinden, werden überschrieben.

jam2 — Der Text wird mit der Farbe in frontPen geschrieben, der Hintergrund erhält die Farbe aus backPen.

complement — Bildpunkte, die sich schon an der Stelle des Textes befinden, werden gelöscht (nicht überschrieben).

inversvid — Der Text wird invers dargestellt, d.h. daß Vorder- und Hintergrundfarbe bei der Ausgabe des Textes vertauscht werden.

specialInfo — Dieses Feld enthält bei Proportional- oder String-Gadgets einen Zeiger auf die specialInfo-Struktur. Wollen Sie ein Boolean-Gadget verwenden, tragen Sie hier bitte NIL ein.

gadgetID — In dieses Feld müssen Sie eine Zahl (am besten eine Konstante) eintragen, damit Sie bei mehreren Gadgets diese bei der Auswertung voneinander unterscheiden können.

userData — Hier kann man die Adresse einer eigenen Routine (für die Auswertung z.B.) unterbringen, um eine (allerdings minimale) Geschwindigkeitssteigerung zu erreichen. Dieses Feld dürfte allerdings nur

für absolut extrem schnelle Action-Spiele oder Jet-Simulationen, wo jede millionstel Sekunde zählt, von Bedeutung sein. Ich habe dieses Feld noch nie gebraucht.

3.4.3 Borders

Bevor wir zur Programmierpraxis schreiten, soll noch ein kurzer Record erklärt werden, nämlich der für den Gadgetrahmen zuständige. Wenn Sie sich die Gadgets in professionellen Programmen ansehen, so stellen Sie fest, daß viele davon umrahmt sind. So eine Umrahmung ist mit der Border-Struktur, deren Adresse im Feld gadgetRender des Gadget-Records gespeichert wird, ganz einfach zu erreichen:

```
Border=RECORD
  leftEdge  : INTEGER      ; x-Koordinate der linken, oberen Ecke des Borders,
                             relativ zum Objekt (hier Gadget)!
  topEdge   : INTEGER      ; y-Koordinate der linken, oberen Ecke des Borders,
                             relativ zum Objekt (hier Gadget)!
  frontPen  : Byte         ; Vordergrundfarbe
  backPen   : Byte         ; Hintergrundfarbe
  drawMode  : DrawModeSet  ; Zeichenmodus
  count     : Byte         ; Anzahl der Eckpunkte
  xy        : ADDRESS      ; Zeiger auf die Tabelle mit den Koordinaten der
                             Eckpunkte
  nextBorder: BorderPtr    ; Zeiger auf den nächsten Border (bei doppelter
                             Umrandung, z.B.)
END;
```

Das einzige Feld, das einer Erklärung bedarf, ist xy: An dieses Feld muß ein Zeiger auf das Array, das die Koordinaten der Eckpunkte enthält, übergeben werden. Die geraden Arrayfelder enthalten die x-, die ungeraden die y-Koordinaten, wobei zu beachten ist, daß die Koordinaten sich stets relativ zur linken, oberen Ecke des Borders beziehen. Der letzte Eckpunkt sollte die Koordinaten des Startpunktes enthalten, will man einen geschlossenen Rahmen (nicht den schon vielzitierten gesprengten) erhalten. Ein Beispiel für einen Rahmen, der ein Gadget mit der Breite von 30 Punkten umgeben soll:

```
...
xyFeld: ARRAY[Ø..9] OF INTEGER;
...
xyFeld[Ø]:= Ø; xyFeld[1]:= Ø; (* 1.Punkt — ( Ø; Ø) *)
xyFeld[2]:=31; xyFeld[3]:= Ø; (* 2.Punkt — (31; Ø) *)
xyFeld[4]:=31; xyFeld[5]:=11; (* 3.Punkt — (31;11) *)
xyFeld[6]:= Ø; xyFeld[7]:=11; (* 4.Punkt — ( Ø;11) *)
xyFeld[8]:= Ø; xyFeld[9]:= Ø; (* 5.Punkt — ( Ø; Ø) *)
...
```

Damit müßte nun auch die Programmierung von Borders, die wir für die folgenden Kapitel brauchen, keine Probleme mehr bereiten.

3.4.4 Boolean-Gadgets

Wir wollen uns jetzt, da wir die elementaren Dinge der Gadgetprogrammierung kennen, mit der Praxis auseinandersetzen. Anfangen wollen wir mit den Boolean-Gadgets, die schon ein wenig erläutert wurden. Erstmal muß man sich klar machen, was für ein Boolean-Gadget man überhaupt braucht, denn es gibt zwei verschiedene Arten:

1. Das normale Boolean-Gadget, dessen Anwahl eine bestimmte Funktion auslöst (eine Festplatte formatiert, ein Musikstück startet, einen Guru auslöst, ...).
2. Das TOGGLESELECT-Gadget, mit dem man zwischen zwei Modi (an/aus, ja/nein, rot/grün, ...) hin- und herschalten kann. Der Ausgangszustand ist stets »aus« was aber mit dem SELECTED-Flag (s.o.) leicht geändert werden kann.

Haben Sie die vorherigen Kapitel aufmerksam verfolgt, so wird Ihnen die Programmierung eines einfachen Boolean-Gadgets keine Probleme bereiten. Da aber ein gut dokumentiertes Programm mehr sagt als tausend Worte, soll ein solches zum besseren Verständnis folgen:

```
MODULE BoolGadgetDemo;

FROM SYSTEM    IMPORT ADDRESS,ADR,LONGSET;
FROM Intuition IMPORT NewWindow,WindowPtr,IDCMPFlags,IDCMPFlagSet,
                      WindowFlags,WindowFlagSet,OpenWindow,CloseWindow,
                      IntuiText,Gadget,boolGadget,GadgetFlags,GadgetFlagSet,
                      ActivationFlags,ActivationFlagSet,Border,ScreenFlags,
                      ScreenFlagSet;
FROM Graphics  IMPORT jam1;
FROM Exec      IMPORT SetSignal;

CONST EndGadget=0;

VAR WindowDaten: NewWindow;
    MeinWindow : WindowPtr;
    MeinText   : IntuiText;
    MeinGadget : Gadget;
    MeinRahmen : Border;
    MeinXY     : ARRAY[0..9] OF INTEGER;

BEGIN
  WITH WindowDaten DO (* Fenster initialisieren — kalter Kaffee *)
    leftEdge:=0; topEdge:=0; width:=640; height:=256;
    detailPen:=0; blockPen:=1; idcmpFlags:=IDCMPFlagSet{gadgetUp};
    flags:=WindowFlagSet{windowSizing,windowDrag,windowDepth};
```

```
    firstGadget:=ADR(MeinGadget); checkMark:=NIL; title:=ADR("Gadget-Window");
    bitMap:=NIL; type:=ScreenFlagSet{wbenchScreen}; minWidth:=400;
    maxWidth:=640; minHeight:=20; maxHeight:=256;
  END;

  WITH MeinRahmen DO (* Rahmen initialisieren — schon besser *)
    leftEdge:=-1; topEdge:=-1; frontPen:=1; backPen:=0; drawMode:=jam1;
    count:=5; xy:=ADR(MeinXY); nextBorder:=NIL;
  END;

  (* Eckpunkte des Rahmens definieren *)
  MeinXY[0]:=0 ; MeinXY[1]:=0 ; MeinXY[2]:=39; MeinXY[3]:=0;
  MeinXY[4]:=39; MeinXY[5]:=11; MeinXY[6]:=0 ; MeinXY[7]:=11;
  MeinXY[8]:=0 ; MeinXY[9]:=0 ;

  WITH MeinText DO (* Text initialisieren *)
    frontPen:=1; backPen:=0; drawMode:=jam1; leftEdge:=1; topEdge:=2;
    nextText:=NIL; iText:=ADR("ENDE");
  END;

  WITH MeinGadget DO (* Gadget initialisieren *)
    nextGadget:=NIL; leftEdge:=300; topEdge:=123; width:=38; height:=10;
    flags:=GadgetFlagSet{}; activation:=ActivationFlagSet{gadgImme-
diate, relVerify};
    gadgetType:=boolGadget; gadgetRender:=ADR(MeinRahmen); selectRender:=NIL;
    gadgetText:=ADR(MeinText); specialInfo:=NIL; gadgetID:=EndGadget;
    userData:=NIL;
  END;

  MeinWindow:=OpenWindow(WindowDaten);
  IF MeinWindow#NIL THEN
    WHILE NOT (MeinWindow^.userPort^.sigBit IN SetSignal(LONGSET{},LONGSET{}))
    DO
    END;
    CloseWindow(MeinWindow);
  END;
END BoolGadgetDemo.
```

3.4.5 String-Gadgets

Die Programmierung von String-Gadgets ist ein wenig komplizierter als die von Boolean-Gadgets, da erstgenannte zusätzliche Informationen benötigen. Diese werden im StringInfo-Record, dessen Adresse dem Feld specialInfo der Gadget-Struktur übergeben wird, festgelegt:

```
StringInfo=RECORD
  buffer     : ADDRESS    ; Adresse des Speicherbereichs, in dem die eingegebenen
                            Zeichen abgelegt werden
  undoBuffer: ADDRESS     ; Adresse des Speicherbereichs, in dem die letzte
                            Eingabe gespeichert wird
  bufferPos : INTEGER     ; Position des Cursors im Textspeicher beim Öffnen des
                            Gadgets
  maxChars  : INTEGER     ; Zahl der maximal einzugebenden Zeichen
  dispPos   : INTEGER     ; Position des ersten Zeichens

; die folgenden Felder werden von Intuition initialisiert und verwaltet

  undoPos   : INTEGER     ; Position des ersten Zeichens im Undo-Speicher
  numChars  : INTEGER     ; Anzahl der momentan im Buffer befindlichen Zeichen
  dispCount : INTEGER     ; Anzahl der Zeichen, die der Klickbereich darstellen
                            kann
  cLeft     : INTEGER     ; Abstand des Klickbereichs zum linken Fensterrand
  cTop      : INTEGER     ; Abstand des Klickbereichs zum oberen Fensterrand
  layerPtr  : LayerPtr    ; Zeiger auf den Layer des Gadgets

; die letzten beiden Felder müssen Sie wieder selbst initialisieren

  longint   : LONGINT     ; eingegebener Wert bei Integer-Gadget

  altKeyMap : KeyMapPtr   ; Zeiger auf alternative Tastaturbelegung
END;
```

Zu den Feldern, die einer näheren Erklärung bedürfen:

buffer — Hier müssen Sie die Adresse des Strings, in dem Sie die Eingabe, die der Benutzer bei Ihrem Gadget macht, gespeichert haben möchten, angeben. Nützlich ist, daß Sie bereits einen Wert angeben können, indem Sie den buffer-String einfach mit Ihrem Startwert füllen.

undoBuffer — Manchmal möchte der Benutzer die vorherige Eingabe, die er machte, zurückholen. Machen Sie es ihm doch komfortabel und richten Sie einen Undo-Speicher ein, in dem die letzte Eingabe automatisch von Intuition gespeichert wird und der Benutzer sie mit einer Tastenkombination (s.u.) zurückholen kann. UndoBuffer muß auf den Undo-String zeigen.

longint — Akzeptieren Sie nur (ganze) Zahlen in Ihrem String-Gadget, so brauchen Sie nur das Flag LONGINT im Feld activationFlags der Gadget-Struktur zu setzen und in diesem Feld Ihre Variable (keinen Zeiger!) mit einem bereits vordefinierten Wert zu übergeben. Diese enthält dann stets die neuesten Benutzer-Eingaben und kann einfach abgefragt werden.

altKeyMap Möchten Sie einen Vokabeltrainer für Französisch programmieren, so empfiehlt es sich, im Eingabefeld für französische Vokabeln auch eine französische Tastaturtabelle zu setzen. Tun Sie dies, in dem Sie unter activationFlags (Gadget-Record) das Flag ALTKEYMAP angeben und übergeben Sie hier einen Zeiger auf Ihre alternative Tastaturtabelle. Die Programmierung eigener KeyMaps würde den Rahmen dieses Buches sprengen.

Noch einige interessante Tastaturfunktionen für String-/Integer-Gadgets:

[←]	:	Cursor ein Zeichen weiter nach links bewegen
[→]	:	Cursor ein Zeichen weiter nach rechts bewegen
[SHIFT] + [←]	:	Sprung an den Anfang des Textes mit dem Cursor
[SHIFT] + [→]	:	Sprung an das Ende des Textes mit dem Cursor
[DEL]	:	Zeichen unter dem Cursor löschen
[Backspace]	:	Zeichen links vom Cursor löschen
[↵]	:	Eingabe im Buffer speichern
[Amiga RECHTS] + [Q]	:	Undo-Funktion, holt die letzte Eingabe zurück
[Amiga RECHTS] + [X]	:	ganzen Text im Gadget löschen

Und zum nächsten Beispielprogramm:

```
MODULE StringGadgetDemo;

FROM SYSTEM     IMPORT ADDRESS,ADR,LONGSET;
FROM Intuition IMPORT NewWindow,OpenWindow,CloseWindow,WindowPtr,IDCMPFlags,
                      IDCMPFlagSet,WindowFlags,WindowFlagSet,Gadget,strGadget,
                      GadgetFlags,GadgetFlagSet,ActivationFlags,ScreenFlagSet,
                      ActivationFlagSet,Border,StringInfo,ScreenFlags;
FROM Graphics   IMPORT jam1;
FROM Exec       IMPORT SetSignal;
FROM Terminal   IMPORT WriteString,WriteLn;

CONST NameGadget    =1;
      VornameGadget=2;

VAR WindowDaten : NewWindow;
    MeinWindow  : WindowPtr;
    StringGadget: ARRAY[Ø..1] OF Gadget;
    Info        : ARRAY[Ø..1] OF StringInfo;
    Rahmen      : Border;
```

```
    xyFeld       : ARRAY[Ø..9] OF INTEGER;
    Buf,UBuf     : ARRAY[Ø..1],[Ø..8Ø] OF CHAR;

BEGIN
  WITH WindowDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256; detailPen:=Ø;
    blockPen:=1; idcmpFlags:=IDCMPFlagSet{closeWindow};
    flags:=WindowFlagSet{windowDrag,windowDepth,windowDrag,windowClose};
    firstGadget:=ADR(StringGadget[Ø]); checkMark:=NIL; bitMap:=NIL;
    title:=ADR("Bitte geben Sie Ihren Vor- und Nachnamen ein!");
    type:=ScreenFlagSet{wbenchScreen};
    minWidth:=4ØØ; maxWidth:=64Ø; minHeight:=2Ø; maxHeight:=256;
  END;

  WITH Rahmen DO
    leftEdge:=-1; topEdge:=-1; frontPen:=1; backPen:=Ø; drawMode:=jam1;
    count:=5; xy:=ADR(xyFeld); nextBorder:=NIL;
  END;

  xyFeld[Ø]:=  Ø; xyFeld[1]:= Ø;
  xyFeld[2]:=191; xyFeld[3]:= Ø;
  xyFeld[4]:=191; xyFeld[5]:=11;
  xyFeld[6]:=  Ø; xyFeld[7]:=11;
  xyFeld[8]:=  Ø; xyFeld[9]:= Ø;

  Buf [Ø]:="Vorname"; Buf [1]:="Nachname";
  UBuf[Ø]:="HOLGER";  UBuf[1]:="GZELLA";

  WITH Info[Ø] DO
    buffer:=ADR(Buf[Ø]); undoBuffer:=ADR(UBuf[Ø]);
    bufferPos:=Ø; maxChars:=8Ø; dispPos:=Ø;
  END;

  WITH Info[1] DO
    buffer:=ADR(Buf[1]); undoBuffer:=ADR(UBuf[1]);
    bufferPos:=Ø; maxChars:=8Ø; dispPos:=Ø;
  END;

  WITH StringGadget[Ø] DO
    nextGadget:=ADR(StringGadget[1]);
    leftEdge:=7Ø; topEdge:=7Ø; width:=19Ø; height:=1Ø;
    flags:=GadgetFlagSet{}; activation:=ActivationFlagSet{gadgImmediate,
                                                          relVerify,
                                                          toggleSelect,
                                                          stringCenter};
    gadgetType:=strGadget;
    gadgetRender:=ADR(Rahmen); selectRender:=NIL;
```

```
    gadgetText:=NIL; specialInfo:=ADR(Info[0]); gadgetID:=VornameGadget;
    userData:=NIL;
  END;

  WITH StringGadget[1] DO
    nextGadget:=NIL;
    leftEdge:=70; topEdge:=90; width:=190; height:=10;
    flags:=GadgetFlagSet{}; activation:=ActivationFlagSet{gadgImmediate,
                                                          relVerify,
                                                          toggleSelect,
                                                          stringCenter};
    gadgetType:=strGadget;
    gadgetRender:=ADR(Rahmen); selectRender:=NIL;
    gadgetText:=NIL; specialInfo:=ADR(Info[1]); gadgetID:=NameGadget;
    userData:=NIL;
  END;

  MeinWindow:=OpenWindow(WindowDaten);
  WHILE NOT (MeinWindow^.userPort^.sigBit IN SetSignal(LONGSET{},LONGSET{})) DO
  END;

  CloseWindow(MeinWindow);
  WriteLn;
  WriteString("Hallo, "); WriteString(Buf[0]);
  WriteString(" "); WriteString(Buf[1]); WriteString("!");
  WriteLn;
END StringGadgetDemo.
```

Damit dürften Ihnen auch die String-Gadgets keine Probleme mehr bereiten.

3.4.6 Proportional-Gadgets

Hiermit kommen wir zur letzten Unterart der Gadgets: die Proportional-Gadgets, vielfach auch Schieberegler genannt. Schieberegler dienen vor allem der komfortablen Eingabe von Zahlen (besonders Farbwerte – denken Sie nur an Preferences), denn man spielt erfahrungsgemäß lieber an einem Schieberegler herum als daß man irgendwelche nichtssagenden Zahlen ausprobiert, um die richtige Farbkombination herauszufinden (ein kräftiges Rasta-Grün, ein unauffälliges Mausgrau, ...).

Auch Proportional-Gadgets kann man in verschiedene Unterarten aufteilen:

1. horizontale (waagrechte), wie sie bei der Farbeinstellung des Preferences-Programms vorkommen,
2. vertikale (senkrechte), wie man sie oft in Dateiauswahlfeldern (Filerequestern) von professionellen Programmen sieht,
3. zweidimensionale, die horizontal sowie vertikal beweglich sind.

Wie auch schon bei den String-Gadgets ist hier eine spezielle Informationsstruktur vonnöten, um die vielen Spezialinformationen zu fassen, nur daß diese hier PropInfo heißt:

```
PropInfo=RECORD
  flags      : PropInfoFlagSet ; Flags des Gadgets
  horizPot   : CARDINAL        ; horizontale Startposition des Reglers beim
                                 initialisieren
  vertPot    : CARDINAL        ; vertikale Startposition des Reglers beim
                                 initialisieren
  horizBody  : CARDINAL        ; Schrittweite des Reglers in x-Richtung
  vertBody   : CARDINAL        ; Schrittweite des Reglers in y-Richtung

; die folgenden Felder werden von Intuition gesetzt

  cWidth     : CARDINAL        ; Breite des Schiebebereichs
  cHeight    : CARDINAL        ; Höhe des Schiebebereichs
  hPotRes    : CARDINAL        ; Reglerinkrement in x-Richtung
  vPotRes    : CARDINAL        ; Reglerinkrement in y-Richtung
  leftBorder: CARDINAL         ; linker Rahmen des Schiebebereichs
  topBorder : CARDINAL         ; oberer Rahmen des Schiebebereichs
END;
```

Hier die genauere Erklärung einiger Felder:

flags — Für dieses Feld sind alle Werte der Menge PropInfoFlagSet gültig:

- autoKnob: Intuition erzeugt automatisch einen Reglerknopf, ein einfaches, ausgefülltes Rechteck.
- freeHoriz: Wenn dieses Flag gesetzt ist, so kann der Benutzer den Reglerknopf horizontal (in x-Richtung) verschieben.
- freeVert: Haben Sie dieses Flag gesetzt, so ist eine Verschiebung des Reglerknopfes in y-Richtung (vertikal) möglich. Für ein zweidimensionales Gadget müssen FREEHORIZ sowie FREEVERT gesetzt sein.
- propBorderless: Ist dieses Flag gesetzt, so hat das Gadget keinen Rahmen.
- pf4–pf7: Reserviert
- knobHit: Dieses Flag wird von Intuition gesetzt, wenn der Benutzer den Reglerknopf angeklickt hat.

horizBody Hier können Sie die Schrittweite des Reglers angeben. Für manche Gadgets (Farbregler, z.B.) sind nur Schrittweiten bis 16 sinnvoll, da der Amiga nur 16 Abstufungen pro Farbwert annehmen kann. Gerechnet wird dabei immer in Bezug auf 65535 (0FFFFH). Um beispielsweise 16 verschiedene Abstufungen mit dem Abstand je 1 einstellen zu können müssen Sie »65535 DIV 16*1 = 4096 (01000H)« angeben.

Hat der Benutzer an dem Regler den Wert verändert, so setzt Intuition praktischerweise das Feld horizPot automatisch auf den neuesten Stand. Mit »einPropInfo horizPot DIV 4096« kann man den internen Wert in den eigenen umwandeln. Die Zahlen gelten für das Beispiel mit dem Farbregler.

vertBody Dieses Feld ist mit horizBody bis auf die Tatsache, daß es sich auf die vertikale Schrittweite bezieht, identisch.

Nun haben wir auch die Proportional-Gadgets abgeschlossen und es ist wieder Zeit für ein Beispielprogramm:

```
MODULE PropGadgetDemo;

FROM SYSTEM    IMPORT ADDRESS,ADR,LONGSET;
FROM Intuition IMPORT NewWindow,WindowPtr,IDCMPFlags,IDCMPFlagSet,
                      WindowFlags,WindowFlagSet,OpenWindow,CloseWindow,
                      IntuiText,Gadget,propGadget,GadgetFlags,GadgetFlagSet,
                      ActivationFlags,ActivationFlagSet,Border,ScreenFlags,
                      ScreenFlagSet,PropInfoFlags,PropInfoFlagSet,PropInfo;
FROM Graphics  IMPORT jam1;
FROM Exec      IMPORT SetSignal;

CONST Schieber=0;

VAR WindowDaten: NewWindow;
    MeinWindow : WindowPtr;
    MeinGadget : Gadget;
    MeinInfo   : PropInfo;
    MeinRahmen : Border;
    MeinXY     : ARRAY[0..9] OF INTEGER;

BEGIN
  WITH WindowDaten DO (* Fenster initialisieren -- kalter Kaffee *)
    leftEdge:=0; topEdge:=0; width:=640; height:=256;
    detailPen:=0; blockPen:=1; idcmpFlags:=IDCMPFlagSet{closeWindow};
    flags:=WindowFlagSet{windowSizing,windowDrag,windowDepth,
           windowClose, activate};
```

```
    firstGadget:=ADR(MeinGadget); checkMark:=NIL; title:=ADR("Gadget-Window");
    bitMap:=NIL; type:=ScreenFlagSet{wbenchScreen}; minWidth:=400;
    maxWidth:=640; minHeight:=20; maxHeight:=256;
  END;

  WITH MeinRahmen DO (* Rahmen initialisieren — schon besser *)
    leftEdge:=-1; topEdge:=-1; frontPen:=1; backPen:=0; drawMode:=jam1;
    count:=5; xy:=ADR(MeinXY); nextBorder:=NIL;
  END;

  (* Eckpunkte des Rahmens definieren *)
  MeinXY[0]:=0 ; MeinXY[1]:=0 ; MeinXY[2]:=39; MeinXY[3]:=0;
  MeinXY[4]:=39; MeinXY[5]:=11; MeinXY[6]:=0 ; MeinXY[7]:=11;
  MeinXY[8]:=0 ; MeinXY[9]:=0 ;

  WITH MeinInfo DO (* Info initialisieren *)
    flags:=PropInfoFlagSet{autoKnob,freeHoriz,freeVert};
    horizPot:=0; vertPot:=0; horizBody:=4096; vertBody:=4096;
  END;

  WITH MeinGadget DO (* Gadget initialisieren *)
    nextGadget:=NIL; leftEdge:=40; topEdge:=60; width:=199; height:=50;
    flags:=GadgetFlagSet{}; activation:=ActivationFlagSet{gadgImmediate,
    relVerify};
    gadgetType:=propGadget; gadgetRender:=ADR(MeinRahmen); selectRender:=NIL;
    gadgetText:=NIL; specialInfo:=ADR(MeinInfo); gadgetID:=Schieber;
    userData:=NIL;
  END;

  MeinWindow:=OpenWindow(WindowDaten);
  IF MeinWindow#NIL THEN
    WHILE NOT (MeinWindow^.userPort^.sigBit IN SetSignal(LONGSET{},LONGSET{}))
    DO
    END;
    CloseWindow(MeinWindow);
  END;
END PropGadgetDemo.
```

3.4.7 Abfrage mehrerer Gadgets

Wir haben zwar eine elegante und kurze Lösung für die Abfrage einer IDCMP-Nachricht gefunden, doch oftmals hat man mehrere Gadgets in einem Programm, die zugleich abgefragt werden müssen. Dafür bietet sich die erste Möglichkeit der IDCMP-Abfrage an (Kapitel 3.4.6). Mit der IntuiMessage-Struktur können wir dann abfragen, um welches Gadget es sich handelt (denn die Gadgets bekommen ja Nummern von Ihnen). Konkret sieht das so aus: Sie fragen in einem LOOP ab, welches IDCMP-Flag

eine Nachricht auslöste. Erhalten Sie GADGETUP (oder GADGETDOWN), so können Sie mit CASE anhand der Nummern die Gadgets (bekommen Sie aus dem Feld iAddress der IntuiMessage-Struktur) erkennen und dementsprechend reagieren. Wie das nun in der Praxis geht, sehen Sie am folgenden Programm:

```
MODULE MultiGadgets;

FROM SYSTEM    IMPORT ADDRESS,ADR,LONGSET;
FROM Intuition IMPORT NewWindow,WindowPtr,IDCMPFlags,IDCMPFlagSet,
                      WindowFlags,WindowFlagSet,OpenWindow,CloseWindow,
                      IntuiText,Gadget,boolGadget,GadgetFlags,GadgetFlagSet,
                      ActivationFlags,ActivationFlagSet,Border,ScreenFlags,
                      ScreenFlagSet,IntuiMessagePtr,DisplayBeep,GadgetPtr;
FROM Graphics  IMPORT jam1;
FROM Exec      IMPORT GetMsg,ReplyMsg,WaitPort;

CONST EndGadget =1;
      BeepGadget=Ø;

VAR WindowDaten: NewWindow;
    MeinWindow : WindowPtr;
    MeinText   : ARRAY[Ø..1] OF IntuiText;
    MeinGadget : ARRAY[Ø..1] OF Gadget;
    MeinRahmen : Border;
    MeinXY     : ARRAY[Ø..9] OF INTEGER;
    IntuiMsg   : IntuiMessagePtr;
    class      : IDCMPFlagSet;
    adr        : GadgetPtr;

BEGIN
  WITH WindowDaten DO (* Fenster initialisieren — kalter Kaffee *)
    leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256;
    detailPen:=Ø; blockPen:=1; idcmpFlags:=IDCMPFlagSet{closeWindow,gadgetUp};
    flags:=WindowFlagSet{windowSizing,windowDrag,windowDepth,windowClose,
    activate};
    firstGadget:=ADR(MeinGadget); checkMark:=NIL; title:=ADR("Gadget-Window");
    bitMap:=NIL; type:=ScreenFlagSet{wbenchScreen}; minWidth:=4ØØ;
    maxWidth:=64Ø; minHeight:=2Ø; maxHeight:=256;
  END;

  WITH MeinRahmen DO (* Rahmen initialisieren — schon besser *)
    leftEdge:=-1; topEdge:=-1; frontPen:=1; backPen:=Ø; drawMode:=jam1;
    count:=5; xy:=ADR(MeinXY); nextBorder:=NIL;
  END;

  (* Eckpunkte des Rahmens definieren *)
  MeinXY[Ø]:=Ø ; MeinXY[1]:=Ø ; MeinXY[2]:=39; MeinXY[3]:=Ø;
```

```
MeinXY[4]:=39; MeinXY[5]:=11; MeinXY[6]:=0 ; MeinXY[7]:=11;
MeinXY[8]:=0 ; MeinXY[9]:=0 ;

WITH MeinText[0] DO
  frontPen:=1; backPen:=0; drawMode:=jam1; leftEdge:=1; topEdge:=2;
  nextText:=NIL; iText:=ADR("BEEP");
END;

WITH MeinText[1] DO
  frontPen:=1; backPen:=0; drawMode:=jam1; leftEdge:=1; topEdge:=2;
  nextText:=NIL; iText:=ADR("ENDE");
END;

WITH MeinGadget[0] DO
  nextGadget:=ADR(MeinGadget[1]);
  leftEdge:=278; topEdge:=123; width:=38; height:=10;
  flags:=GadgetFlagSet{}; activation:=ActivationFlagSet{gadgImmediate,
  relVerify};
  gadgetType:=boolGadget; gadgetRender:=ADR(MeinRahmen); selectRender:=NIL;
  gadgetText:=ADR(MeinText[0]); specialInfo:=NIL; gadgetID:=BeepGadget;
  userData:=NIL;
END;

WITH MeinGadget[1] DO
  nextGadget:=NIL; leftEdge:=322; topEdge:=123; width:=38; height:=10;
  flags:=GadgetFlagSet{}; activation:=ActivationFlagSet{gadgImmediate,
  relVerify};
  gadgetType:=boolGadget; gadgetRender:=ADR(MeinRahmen); selectRender:=NIL;
  gadgetText:=ADR(MeinText[1]); specialInfo:=NIL; gadgetID:=EndGadget;
  userData:=NIL;
END;

MeinWindow:=OpenWindow(WindowDaten);
IF MeinWindow#NIL THEN
  LOOP
    IntuiMsg:=GetMsg(MeinWindow^.userPort);
    WHILE IntuiMsg#NIL DO
      class:=IntuiMsg^.class; adr:=IntuiMsg^.iAddress;
      ReplyMsg(IntuiMsg);
      IF (closeWindow IN class) THEN EXIT;
      ELSIF (gadgetUp IN class) THEN
        CASE adr^.gadgetID OF
          BeepGadget: DisplayBeep(MeinWindow^.wScreen)
          EndGadget : EXIT
        END;
      END;
```

```
        IntuiMsg:=GetMsg(MeinWindow^.userPort);
      END;
    END;
    CloseWindow(MeinWindow);
  END;
END MultiGadgets.
```

Sollten Sie dieses Programm nicht verstanden haben, so ist das nicht weiter schlimm, hier folgt eine Erklärung:

Nach der ganzen Initialisierung durchläuft das Programm eine Endlosschleife (den LOOP). Dann wird eine Message geholt, und sollte deren Inhalt sinnvoll sein (ein Zeiger existiert dann darauf) wird sie in einer WHILE-Schleife ausgewertet. Die Art der Message und die Adresse (in der in diesem Fall ein Zeiger auf das angewählte Gadget stehen müßte) werden dann in eigene Variablen gerettet, da sie beim Aufruf von REPLYMSG verloren gehen. Nun kommt die eigentliche Abfrage: Wurde das Schließsymbol angeklickt (CLOSEWINDOW ist gesetzt), wird die Endlosschleife unterbrochen und das Window geschlossen. Hat der Benutzer aber eines der beiden Boolean-Gadgets angeklickt (GADGETUP ist gesetzt), wird dieses mit einer CASE-Auswertung spezifiert (im Feld gadgetID steht ja die vom Programmierer zugewiesene Nummer). Beim BEEP-Gadget blinkt der Bildschirm kurz auf, beim ENDE-Gadget geschieht das gleiche wie bei der Anwahl des Schließsymbols. Dann wird eine neue Message geholt und die Auswertung beginnt von neuem.

3.4.8 Gadget-Befehle

Für die komfortable Gadget-Behandlung gibt es einige Befehle, die im folgenden besprochen werden sollen.

Kommt der Wunsch auf, ein Gadget nachträglich an die interne Liste anzuhängen, bediene man sich des ADDGADGET-Befehls:

RealPos := AddGadget (MeinWindow , MeinGadget , MeinePosition);

MeinWindow	WindowPtr auf das Fenster, an dessen Liste das Gadget angehängt werden soll.
MeinGadget	GadgetPtr auf das Gadget, das angehängt werden soll.
MeinePosition	INTEGER-Wert der Position, an der das Gadget angehängt werden soll.
RealPos	INTEGER-Wert der Position, an der Intuition das Gadget angehängt hat.

Im umgekehrten Fall, ein Gadget soll entfernt werden, kann man den REMOVE-GADGET-Befehl nutzen:

AltePos := RemoveGadget (MeinWindow , MeinGadget);

MeinWindow	WindowPtr auf das Window, aus dessen Liste ein Gadget entfernt werden soll.
MeinGadget	GadgetPtr auf das Gadget, das zu entfernen ist.
AltePos	INTEGER-Wert der Position, die das Gadget vor dem Entfernen hatte.

Doch manchmal ist es sinnvoll (wenn nicht gar notwendig), Gadgets ein- und auszuschalten. Ein ausgeschaltetes Gadget wird »ghosted«, also punktiert (verschwommen) gezeichnet. Um diesen Zustand auch bei eigenen Gadgets zu erreichen, gibt es die Befehle OFFGADGET und ONGADGET:

OffGadget (MeinGadget , MeinWindow , MeinRequester);

MeinGadget	GadgetPtr auf das Gadget, das ausgeschaltet werden soll.
MeinWindow	WindowPtr auf das Window, in dem sich das auszuschaltende Gadget befindet.
MeinRequester	RequesterPtr auf den Requester, in dem sich das Gadget befindet, falls ein solcher existiert. Ist das Gadget einfach nur im Fenster und nicht im Requester, ist hier NIL anzugeben.

Nun der Befehl, um ein Gadget wieder einzuschalten:

OnGadget (MeinGadget , MeinWindow , MeinRequester);

MeinGadget	GadgetPtr auf das (ausgeschaltete) Gadget, das wieder eingeschaltet werden soll.
MeinWindow	WindowPtr auf das Window, in dem sich das Gadget befindet.
MeinRequester	RequesterPtr auf den Requester, in dem sich das Gadget befindet. Sollte das Gadget einfach nur im Fenster befindlich sein, so geben Sie hier am besten NIL an.

Sollen die Gadgets eines Fensters neu gezeichnet (»refresht«) werden, so empfehle ich, den REFRESHGADGETS-Befehl zu verwenden:

RefreshGadgets (MeinGadget , MeinWindow , MeinRequester);

MeinGadget	GadgetPtr auf das Gadget, bei dem die Erneuerung beginnen soll.
MeinWindow	WindowPtr auf das Window, in dem sich die zu erneuernden Gadgets befinden.
MeinRequester	RequesterPtr auf den Requester, dessen Gadgets erneuert werden sollen, falls ein solcher existiert. Sollen nur die Gadgets im Fenster erneuert werden, ist hier NIL anzugeben.

Kommen wir nun zum letzten und komplexesten Gadget-Befehl, nämlich MODIFY-PROP. MODIFYPROP erlaubt die Modifizierung eines bereits auf dem Bildschirm befindlichen Schiebereglers:

ModifyProp (MeinGadget , MeinWindow , MeinRequester , MeineFlags , MeinHorizPot , MeinVertPos , MeinHorizBody , MeinVertBody);

MeinGadget	GadgetPtr auf das Proportional-Gadget, das modifiziert werden soll.
MeinWindow	WindowPtr auf das Fenster, in dem sich das Gadget befindet.
MeinRequester	Sollte sich das Gadget in einem Requester befinden, enthält dieser Parameter einen RequesterPtr darauf, andernfalls NIL.
MeineFlags	PropInfoFlagSet mit den neuen Flags. Siehe dazu das entsprechende Kapitel.
MeinHorizPot	CARDINAL-Wert der neuen horizontalen Schrittweite.
MeinVertPot	CARDINAL-Wert der neuen vertikalen Schrittweite.
MeinHorizBody	CARDINAL-Wert, der die neue horizontale Initialposition des Reglerknopfes angibt.
MeinVertBody	CARDINAL-Wert, der die neue vertikale Initialposition des Reglerknopfes angibt.

Die Parameter sind identisch mit denen, die in der PropInfo-Struktur angegeben werden.

3.4.9 Ein komfortabler Compiler-Driver

Da wir nun am Ende des Gadget-Kapitels angelangt sind, möchte ich Ihnen ein etwas größeres Demoprogramm, das zur Festigung des hier erworbenen Wissens und gleichzeitig als Demonstration einiger DOS-Befehle dienen soll, präsentieren. Es handelt sich um einen komfortablen Compiler-Driver, mit dem die wichtigsten Kompilierbefehle per Maus durchgeführt werden können. Gerade bei langen Programmnamen wie

»Donaudampfschiffahrtkapitänspatenttestprogramminklusivewunderbarergrafik-undsound.mod«

(zugegeben, das war ein wenig übertrieben) zehrt es sehr an den Nerven, nach der Berichtigung einiger Fehler (was bei größeren Programmen mehrmals der Fall ist) den ganzen Kompilier- und Link-Kontext nochmal einzutippen, da die Tippfehlerhäufigkeit bei langen komplizierten Namen zunimmt. Nach dem Start des Drivers erscheint ein Window mit 17 Gadgets auf dem Bildschirm, die folgende Funktionen haben:

EDIT: Editiert ein Programm mit dem in NAME spezifizierten Namen (die entsprechende Endung wird automatisch angehängt).

COMPILE: Kompiliert das Programm mit dem in NAME eingegebenen Namen (auch hier wird die entsprechende Endung angehängt).

LINK: Linkt den Objektcode das Programms, dessen Namen in NAME angegeben wurde (die entsprechende Endung wird angehängt).

RUN: Startet das Programm mit dem Namen, der in NAME steht.

DO: Kompiliert und linkt das in NAME angegebene Programm unter Berücksichtigung der entsprechenden Endungen.

BATCH!: Für eine Batch-Datei aus (äquivalent zum CLI-Befehl Execute), deren Name in NAME angegeben ist.

LOAD: Lädt die unter OPTIONS spezifizierten Optionen (Compilername, Compileroptionen, ...).

SAVE: Speichert die augenblicklichen Optionen (Inhalte der String-Gadgets) unter dem in OPTIONS angegebenen Namen.

NAME: Legt den Namen für die Kommando-Gadgets EDIT, COMPILE, LINK, DO, RUN und BATCH! fest. Der Name sollte ohne (!) Endung angegeben werden.

C.-OPT.: Hier werden die Optionen für den Kompiliervorgang eingegeben (interessant für C-Compiler).

L.-OPT.: Geben Sie in diesem Feld die Optionen für den Linker ein (genau das richtige für Lattice-C-Benutzer).

EDITOR: In diesem String-Gadget wird der Editor spezifiziert (M2Emacs für M2Amiga, M2Ed für Benchmark, ...).

COMPILER: Definieren Sie hier Ihren Compiler (M2C, M2, LC, CC, ...).

LINKER: Dieses Feld ist für den Namen Ihres Linker da (M2L, M2LK, ALINK, BLINK, ...).

OPTIONS: In diesem Feld wird der Name der Optionsdatei, in der alle eingegebenen Optionen mit SAVE gespeichert werden können, angegeben.

SRC-END.: Dient zur Eingabe der Endung für die Quelldatei.

OBJ-END.: Da verschiedene Linker unterschiedliche Endungen für die Objektdateien brauchen, entstand dieses Feld.

Standardmäßig ist dieses Programm für den M2Amiga-Compiler angepaßt, was zu ändern aber mittels einer Optionsdatei, die Sie auf Ihre Arbeitsdiskette speichern können, kein Problem sein dürfte. Natürlich können Sie das Programm auch für Ihren Compiler anpassen und die Änderungen direkt im Programm vornehmen. Benchmark-Besitzer finden im selben Verzeichnis noch ein Optionsfile namens »Benchmark OPT« für ihren Compiler. Beachten Sie bitte, daß sich alle Dateinamen auf das mit CD spezifizierte Laufwerk beziehen.

Bleibt noch zu sagen, daß die Idee zu diesem Driver dem Programm »Command« von Stefan Diestelmann aus der Amiga 3/88 entstammt. Dieses ist leider in C geschrieben und etwas unflexibel (nur auf den Aztec-C-Compiler beschränkt), doch nun soll der Quellcode von DRIVER v1.00 folgen:

```
(* Ein Compiler-Driver von Holger Gzella / 16.10.88 *)
MODULE Driver;

FROM SYSTEM     IMPORT ADR,ADDRESS;
FROM Intuition  IMPORT NewWindow,OpenWindow,CloseWindow,WindowPtr,GadgetPtr,
                       boolGadget,strGadget,IntuiMessagePtr,IDCMPFlags,
                       IDCMPFlagSet,WindowFlags,WindowFlagSet,GadgetFlags,
                       GadgetFlagSet,ActivationFlags,ActivationFlagSet,
                       Border,StringInfo,IntuiText,Gadget,ScreenFlagSet,
                       ScreenFlags,RefreshGadgets;
FROM Graphics   IMPORT jam1,jam2;
```

```
FROM Dos        IMPORT Execute,FileHandlePtr,Read,Write,Open,Close,oldFile,
                       newFile;
FROM Exec       IMPORT WaitPort,GetMsg,ReplyMsg;
FROM Strings    IMPORT Insert,Delete,Length;

(* Definition der Gadget-Nummern *)

CONST Edit       =  1;
      Compile    =  2;
      Link       =  3;
      Run        =  4;
      Do         =  5;
      Batch      =  6;
      Name       =  7;
      Copt       =  8;
      Lopt       =  9;
      Editor     = 10;
      Compiler   = 11;
      Linker     = 12;
      Options    = 13;
      SEnding    = 14;
      OEnding    = 15;
      Load       = 16;
      Save       = 17;

      ExecString = "Execute ";
      Space      = " ";

(* Variablendefinition *)

VAR
      MeinWindow : WindowPtr;
      WindowDaten: NewWindow;
      MeinGadget : ARRAY[Edit..Save]     OF Gadget;
      MeinInfo   : ARRAY[0..8]           OF StringInfo;
      MeinText   : ARRAY[1..17]          OF IntuiText;
      MeinBorder : ARRAY[0..5]           OF Border;
      MeinXY     : ARRAY[0..6],[0..9]    OF INTEGER;
      Buf,UBuf   : ARRAY[0..6],[0..100]  OF CHAR;
      EBuf,EUBuf : ARRAY[0..1],[0..10]   OF CHAR;
      IntuiMsg   : IntuiMessagePtr;
      class      : IDCMPFlagSet;
      WhichGad   : GadgetPtr;
      i          : INTEGER;
      DoItString : ARRAY[0..150]         OF CHAR;
      ok         : LONGINT;
      MeinFile   : FileHandlePtr;
```

```
PROCEDURE InitAll;
  BEGIN
    Buf[3]:="m2emacs";
    Buf[4]:="m2c";
    Buf[5]:="m2l";
    Buf[6]:="myOptions";
    EBuf[0]:=".mod";
    EBuf[1]:=".obj";

    WITH WindowDaten DO
      leftEdge:=0; topEdge:=0; width:=361; height:=175; detailPen:=0;
      blockPen:=1; idcmpFlags:=IDCMPFlagSet{closeWindow,gadgetUp};
      flags:=WindowFlagSet{windowClose,windowDrag,windowDepth,windowClose,
      activate}; firstGadget:=ADR(MeinGadget[Edit]); checkMark:=NIL;
      title:=ADR("Compiler-Driver by Holger Gzella"); bitMap:=NIL;
      minWidth:=0; maxWidth:=0; minHeight:=0; maxHeight:=0;
      type:=ScreenFlagSet{wbenchScreen};
    END;

    MeinXY[0][0]:=0; MeinXY[0][1]:=0; MeinXY[0][2]:=37; MeinXY[0][3]:=0;
    MeinXY[0][4]:=37; MeinXY[0][5]:=11; MeinXY[0][6]:=0; MeinXY[0][7]:=11;
    MeinXY[0][8]:=0; MeinXY[0][9]:=1;

    MeinXY[1][0]:=0; MeinXY[1][1]:=0; MeinXY[1][2]:=61; MeinXY[1][3]:=0;
    MeinXY[1][4]:=61; MeinXY[1][5]:=11; MeinXY[1][6]:=0; MeinXY[1][7]:=11;
    MeinXY[1][8]:=0; MeinXY[1][9]:=1;

    MeinXY[2][0]:=0; MeinXY[2][1]:=0; MeinXY[2][2]:=29; MeinXY[2][3]:=0;
    MeinXY[2][4]:=29; MeinXY[2][5]:=11; MeinXY[2][6]:=0; MeinXY[2][7]:=11;
    MeinXY[2][8]:=0; MeinXY[2][9]:=1;

    MeinXY[3][0]:=0; MeinXY[3][1]:=0; MeinXY[3][2]:=53; MeinXY[3][3]:=0;
    MeinXY[3][4]:=53; MeinXY[3][5]:=11; MeinXY[3][6]:=0; MeinXY[3][7]:=11;
    MeinXY[3][8]:=0; MeinXY[3][9]:=1;

    MeinXY[4][0]:=0; MeinXY[4][1]:=0; MeinXY[4][2]:=282; MeinXY[4][3]:=0;
    MeinXY[4][4]:=282; MeinXY[4][5]:=11; MeinXY[4][6]:=0; MeinXY[4][7]:=11;
    MeinXY[4][8]:=0; MeinXY[4][9]:=1;

    MeinXY[5][0]:=0; MeinXY[5][1]:=0; MeinXY[5][2]:=21; MeinXY[5][3]:=0;
    MeinXY[5][4]:=21; MeinXY[5][5]:=11; MeinXY[5][6]:=0; MeinXY[5][7]:=11;
    MeinXY[5][8]:=0; MeinXY[5][9]:=1;

    FOR i:=0 TO 5 DO
      WITH MeinBorder[i] DO
        leftEdge:=-3; topEdge:=-2; frontPen:=1; backPen:=0; drawMode:=jam1;
        count:=5; xy:=ADR(MeinXY[i]);
      END;
    END;
```

```
WITH MeinText[1] DO
  frontPen:=3; backPen:=2; drawMode:=jam2; leftEdge:=0; topEdge:=0;
  nextText:=NIL; iText:=ADR("EDIT");
END;

WITH MeinText[2] DO
  frontPen:=3; backPen:=2; drawMode:=jam2; leftEdge:=0; topEdge:=0;
  nextText:=NIL; iText:=ADR("COMPILE");
END;

WITH MeinText[3] DO
  frontPen:=3; backPen:=2; drawMode:=jam2; leftEdge:=0; topEdge:=0;
  nextText:=NIL; iText:=ADR("LINK");
END;

WITH MeinText[4] DO
  frontPen:=3; backPen:=2; drawMode:=jam2; leftEdge:=0; topEdge:=0;
  nextText:=NIL; iText:=ADR("RUN");
END;

WITH MeinText[5] DO
  frontPen:=3; backPen:=2; drawMode:=jam2; leftEdge:=0; topEdge:=0;
  nextText:=NIL; iText:=ADR("DO");
END;

WITH MeinText[6] DO
  frontPen:=3; backPen:=2; drawMode:=jam2; leftEdge:=0; topEdge:=0;
  nextText:=NIL; iText:=ADR("BATCH!");
END;

WITH MeinText[7] DO
  frontPen:=3; backPen:=2; drawMode:=jam2; leftEdge:=0; topEdge:=0;
  nextText:=NIL; iText:=ADR("LOAD");
END;

WITH MeinText[8] DO
  frontPen:=3; backPen:=2; drawMode:=jam2; leftEdge:=0; topEdge:=0;
  nextText:=NIL; iText:=ADR("SAVE");
END;

WITH MeinText[9] DO
  frontPen:=3; backPen:=0; drawMode:=jam1; leftEdge:=-69; topEdge:=0;
  nextText:=NIL; iText:=ADR("NAME    ");
END;

WITH MeinText[10] DO
  frontPen:=3; backPen:=0; drawMode:=jam1; leftEdge:=-69; topEdge:=0;
  nextText:=NIL; iText:=ADR("C.-Opt. ");
END;
```

```
WITH MeinText[11] DO
  frontPen:=3; backPen:=Ø; drawMode:=jam1; leftEdge:=-69; topEdge:=Ø;
  nextText:=NIL; iText:=ADR("L.-Opt. ");
END;

WITH MeinText[12] DO
  frontPen:=3; backPen:=Ø; drawMode:=jam1; leftEdge:=-69; topEdge:=Ø;
  nextText:=NIL; iText:=ADR("Editor  ");
END;

WITH MeinText[13] DO
  frontPen:=3; backPen:=Ø; drawMode:=jam1; leftEdge:=-69; topEdge:=Ø;
  nextText:=NIL; iText:=ADR("Compiler");
END;

WITH MeinText[14] DO
  frontPen:=3; backPen:=Ø; drawMode:=jam1; leftEdge:=-69; topEdge:=Ø;
  nextText:=NIL; iText:=ADR("Linker  ");
END;

WITH MeinText[15] DO
  frontPen:=3; backPen:=Ø; drawMode:=jam1; leftEdge:=-69; topEdge:=Ø;
  nextText:=NIL; iText:=ADR("Options ");
END;

WITH MeinText[16] DO
  frontPen:=3; backPen:=Ø; drawMode:=jam1; leftEdge:=-69; topEdge:=Ø;
  nextText:=NIL; iText:=ADR("Src-End.");
END;

WITH MeinText[17] DO
  frontPen:=3; backPen:=Ø; drawMode:=jam1; leftEdge:=-69; topEdge:=Ø;
  nextText:=NIL; iText:=ADR("Obj-End.");
END;

WITH MeinInfo[Ø] DO
  buffer:=ADR(Buf[Ø]); undoBuffer:=ADR(UBuf[Ø]); bufferPos:=Ø;
  maxChars:=35; dispPos:=Ø;
END;

WITH MeinInfo[1] DO
  buffer:=ADR(Buf[1]); undoBuffer:=ADR(UBuf[1]); bufferPos:=Ø;
  maxChars:=35; dispPos:=Ø;
END;
```

```
WITH MeinInfo[2] DO
  buffer:=ADR(Buf[2]); undoBuffer:=ADR(UBuf[2]); bufferPos:=Ø;
  maxChars:=35; dispPos:=Ø;
END;

WITH MeinInfo[3] DO
  buffer:=ADR(Buf[3]); undoBuffer:=ADR(UBuf[3]); bufferPos:=Ø;
  maxChars:=35; dispPos:=Ø;
END;

WITH MeinInfo[4] DO
  buffer:=ADR(Buf[4]); undoBuffer:=ADR(UBuf[4]); bufferPos:=Ø;
  maxChars:=35; dispPos:=Ø;
END;

WITH MeinInfo[5] DO
  buffer:=ADR(Buf[5]); undoBuffer:=ADR(UBuf[5]); bufferPos:=Ø;
  maxChars:=35; dispPos:=Ø;
END;

WITH MeinInfo[6] DO
  buffer:=ADR(Buf[6]); undoBuffer:=ADR(UBuf[6]); bufferPos:=Ø;
  maxChars:=35; dispPos:=Ø;
END;

WITH MeinInfo[7] DO
  buffer:=ADR(EBuf[Ø]); undoBuffer:=ADR(EUBuf[Ø]); bufferPos:=Ø;
  maxChars:=4; dispPos:=Ø;
END;

WITH MeinInfo[8] DO
  buffer:=ADR(EBuf[1]); undoBuffer:=ADR(EUBuf[1]); bufferPos:=Ø;
  maxChars:=4; dispPos:=Ø;
END;

WITH MeinGadget[Edit] DO
  nextGadget:=ADR(MeinGadget[Compile]); leftEdge:=9; topEdge:=14;
  width:=32; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=boolGadget; gadgetRender:=ADR(MeinBorder[Ø]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[1]); specialInfo:=NIL;
  gadgetID:=Edit; userData:=NIL;
END;

WITH MeinGadget[Compile] DO
  nextGadget:=ADR(MeinGadget[Link]); leftEdge:=51; topEdge:=14;
  width:=56; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
```

```
  gadgetType:=boolGadget; gadgetRender:=ADR(MeinBorder[1]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[2]); specialInfo:=NIL;
  gadgetID:=Compile; userData:=NIL;
END;

WITH MeinGadget[Link] DO
  nextGadget:=ADR(MeinGadget[Run]); leftEdge:=117; topEdge:=14;
  width:=32; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=boolGadget; gadgetRender:=ADR(MeinBorder[Ø]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[3]); specialInfo:=NIL;
  gadgetID:=Link; userData:=NIL;
END;

WITH MeinGadget[Run] DO
  nextGadget:=ADR(MeinGadget[Do]); leftEdge:=159; topEdge:=14;
  width:=24; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=boolGadget; gadgetRender:=ADR(MeinBorder[2]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[4]); specialInfo:=NIL;
  gadgetID:=Run; userData:=NIL;
END;

WITH MeinGadget[Do] DO
  nextGadget:=ADR(MeinGadget[Batch]); leftEdge:=193; topEdge:=14;
  width:=16; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=boolGadget; gadgetRender:=ADR(MeinBorder[5]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[5]); specialInfo:=NIL;
  gadgetID:=Do; userData:=NIL;
END;

WITH MeinGadget[Batch] DO
  nextGadget:=ADR(MeinGadget[Name]); leftEdge:=219; topEdge:=14;
  width:=48; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=boolGadget; gadgetRender:=ADR(MeinBorder[3]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[6]); specialInfo:=NIL;
  gadgetID:=Batch; userData:=NIL;
END;

WITH MeinGadget[Name] DO
  nextGadget:=ADR(MeinGadget[Copt]); leftEdge:=74; topEdge:=30;
  width:=277; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=strGadget; gadgetRender:=ADR(MeinBorder[4]);
```

```
  selectRender:=NIL; gadgetText:=ADR(MeinText[9]);
  specialInfo:=ADR(MeinInfo[0]); gadgetID:=Name; userData:=NIL;
END;

WITH MeinGadget[Copt] DO
  nextGadget:=ADR(MeinGadget[Lopt]); leftEdge:=74; topEdge:=46;
  width:=277; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=strGadget; gadgetRender:=ADR(MeinBorder[4]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[10]);
  specialInfo:=ADR(MeinInfo[1]); gadgetID:=Copt; userData:=NIL;
END;

WITH MeinGadget[Lopt] DO
  nextGadget:=ADR(MeinGadget[Editor]); leftEdge:=74; topEdge:=62;
  width:=277; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=strGadget; gadgetRender:=ADR(MeinBorder[4]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[11]);
  specialInfo:=ADR(MeinInfo[2]); gadgetID:=Lopt; userData:=NIL;
END;

WITH MeinGadget[Editor] DO
  nextGadget:=ADR(MeinGadget[Compiler]); leftEdge:=74; topEdge:=78;
  width:=277; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=strGadget; gadgetRender:=ADR(MeinBorder[4]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[12]);
  specialInfo:=ADR(MeinInfo[3]); gadgetID:=Editor; userData:=NIL;
END;

WITH MeinGadget[Compiler] DO
  nextGadget:=ADR(MeinGadget[Linker]); leftEdge:=74; topEdge:=94;
  width:=277; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=strGadget; gadgetRender:=ADR(MeinBorder[4]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[13]);
  specialInfo:=ADR(MeinInfo[4]); gadgetID:=Compiler; userData:=NIL;
END;

WITH MeinGadget[Linker] DO
  nextGadget:=ADR(MeinGadget[Options]); leftEdge:=74; topEdge:=110;
  width:=277; height:=8; flags:=GadgetFlagSet{};
  activation:=ActivationFlagSet{relVerify,gadgImmediate};
  gadgetType:=strGadget; gadgetRender:=ADR(MeinBorder[4]);
  selectRender:=NIL; gadgetText:=ADR(MeinText[14]);
  specialInfo:=ADR(MeinInfo[5]); gadgetID:=Linker; userData:=NIL;
END;
```

```
  WITH MeinGadget[Options] DO
    nextGadget:=ADR(MeinGadget[SEnding]); leftEdge:=74; topEdge:=126;
    width:=277; height:=8; flags:=GadgetFlagSet{};
    activation:=ActivationFlagSet{relVerify,gadgImmediate};
    gadgetType:=strGadget; gadgetRender:=ADR(MeinBorder[4]);
    selectRender:=NIL; gadgetText:=ADR(MeinText[15]);
    specialInfo:=ADR(MeinInfo[6]); gadgetID:=Options; userData:=NIL;
  END;

  WITH MeinGadget[SEnding] DO
    nextGadget:=ADR(MeinGadget[OEnding]); leftEdge:=74; topEdge:=142;
    width:=37; height:=8; flags:=GadgetFlagSet{};
    activation:=ActivationFlagSet{relVerify,gadgImmediate};
    gadgetType:=strGadget; gadgetRender:=ADR(MeinBorder[Ø]);
    selectRender:=NIL; gadgetText:=ADR(MeinText[16]);
    specialInfo:=ADR(MeinInfo[7]); gadgetID:=SEnding; userData:=NIL;
  END;

 WITH MeinGadget[OEnding] DO
    nextGadget:=ADR(MeinGadget[Load]); leftEdge:=74; topEdge:=158;
    width:=37; height:=8; flags:=GadgetFlagSet{};
    activation:=ActivationFlagSet{relVerify,gadgImmediate};
    gadgetType:=strGadget; gadgetRender:=ADR(MeinBorder[Ø]);
    selectRender:=NIL; gadgetText:=ADR(MeinText[17]);
    specialInfo:=ADR(MeinInfo[8]); gadgetID:=OEnding; userData:=NIL;
  END;

  WITH MeinGadget[Load] DO
    nextGadget:=ADR(MeinGadget[Save]); leftEdge:=277; topEdge:=14;
    width:=32; height:=8; flags:=GadgetFlagSet{};
    activation:=ActivationFlagSet{relVerify,gadgImmediate};
    gadgetType:=boolGadget; gadgetRender:=ADR(MeinBorder[Ø]);
    selectRender:=NIL; gadgetText:=ADR(MeinText[7]); specialInfo:=NIL;
    gadgetID:=Load; userData:=NIL;
  END;

  WITH MeinGadget[Save] DO
    nextGadget:=NIL; leftEdge:=319; topEdge:=14;
    width:=32; height:=8; flags:=GadgetFlagSet{};
    activation:=ActivationFlagSet{relVerify,gadgImmediate};
    gadgetType:=boolGadget; gadgetRender:=ADR(MeinBorder[Ø]);
    selectRender:=NIL; gadgetText:=ADR(MeinText[8]); specialInfo:=NIL;
    gadgetID:=Save; userData:=NIL;
  END;
END InitAll;
```

```
PROCEDURE DoEdit;
  BEGIN
    Insert(DoItString,Ø,EBuf[Ø]);
    Insert(DoItString,Ø,Buf[Ø]);
    Insert(DoItString,Ø,Space);
    Insert(DoItString,Ø,Buf[3]);
    ok:=Execute(ADR(DoItString),NIL,NIL);
    Delete(DoItString,Ø,Length(DoItString));
  END DoEdit;

PROCEDURE DoCompile;
  BEGIN
    Insert(DoItString,Ø,Buf[1]);
    Insert(DoItString,Ø,EBuf[Ø]);
    Insert(DoItString,Ø,Buf[Ø]);
    Insert(DoItString,Ø,Space);
    Insert(DoItString,Ø,Buf[4]);
    ok:=Execute(ADR(DoItString),NIL,NIL);
    Delete(DoItString,Ø,Length(DoItString));
  END DoCompile;

PROCEDURE DoLink;
  BEGIN
    Insert(DoItString,Ø,Buf[2]);
    Insert(DoItString,Ø,EBuf[1]);
    Insert(DoItString,Ø,Buf[Ø]);
    Insert(DoItString,Ø,Space);
    Insert(DoItString,Ø,Buf[5]);
    ok:=Execute(ADR(DoItString),NIL,NIL);
    Delete(DoItString,Ø,Length(DoItString));
  END DoLink;

PROCEDURE DoRun;
  BEGIN
    Insert(DoItString,Ø,Buf[Ø]);
    ok:=Execute(ADR(DoItString),NIL,NIL);
    Delete(DoItString,Ø,Length(DoItString));
  END DoRun;

PROCEDURE DoDo;
  BEGIN
    DoCompile;
    DoLink;
  END DoDo;
```

```
PROCEDURE DoBatch;
  BEGIN
    Insert(DoItString,Ø,Buf[Ø]);
    Insert(DoItString,Ø,ExecString);
    ok:=Execute(ADR(DoItString),NIL,NIL);
    Delete(DoItString,Ø,Length(DoItString));
  END DoBatch;

PROCEDURE DoLoad;
  BEGIN
    MeinFile:=Open(ADR(Buf[6]),oldFile);
    FOR i:=Ø TO 6 DO
      ok:=Read(MeinFile,ADR(Buf[i]),SIZE(Buf[i]));
    END;
    ok:=Read(MeinFile,ADR(EBuf[Ø]),SIZE(EBuf[Ø]));
    ok:=Read(MeinFile,ADR(EBuf[1]),SIZE(EBuf[1]));
    RefreshGadgets(ADR(MeinGadget[Name]),MeinWindow,NIL);
    Close(MeinFile);
  END DoLoad;

PROCEDURE DoSave;
  BEGIN
    MeinFile:=Open(ADR(Buf[6]),newFile);
    FOR i:=Ø TO 6 DO
      ok:=Write(MeinFile,ADR(Buf[i]),SIZE(Buf[i]));
    END;
    ok:=Write(MeinFile,ADR(EBuf[Ø]),SIZE(EBuf[Ø]));
    ok:=Write(MeinFile,ADR(EBuf[1]),SIZE(EBuf[1]));
    Close(MeinFile);
  END DoSave;

PROCEDURE GadgetProg;
  BEGIN
    CASE WhichGad^.gadgetID OF
      Edit   : DoEdit|
      Compile: DoCompile|
      Link   : DoLink|
      Run    : DoRun|
      Do     : DoDo|
      Batch  : DoBatch|
      Load   : DoLoad|
      Save   : DoSave
    ELSE
    END;
  END GadgetProg;
```

```
BEGIN
  InitAll;
  MeinWindow:=OpenWindow(WindowDaten);

  LOOP
    WaitPort(MeinWindow^.userPort);
    IntuiMsg:=GetMsg(MeinWindow^.userPort);
    WHILE IntuiMsg#NIL DO
      class:=IntuiMsg^.class;
      WhichGad:=IntuiMsg^.iAddress;
      ReplyMsg(IntuiMsg);
      IF (closeWindow IN class) THEN EXIT;
      ELSE GadgetProg;
      END;
      IntuiMsg:=GetMsg(MeinWindow^.userPort);
    END;
  END;

  CloseWindow(MeinWindow);
END Driver.
```

3.5 Menüs

Pull-down-Menüs sind sicher wesentliche Elemente der komfortablen Programmbedienung. Beim C64 gab es ja schon so einige Tricks mit »Pseudo-Smartrefresh-Windows« und Pull-down-Menüs, die mit der CTRL-Taste aktiviert werden konnten (im 64er-Sonderheft 12 kann man ein ebenso schönes wie kompliziertes Maschinenprogramm von Said Baloui finden), doch die Art, wie Intuition die Programmierung dieser verhältnismäßig komplexen Materie unterstützt, sprengt anscheinend alle Rahmen. Lassen Sie uns also beginnen.

3.5.1 Menüprogrammierung

Wie Sie sicher schon beobachtet haben, hat jedes Menü einen Oberbegriff (z.B. »Workbench«), unter dem sich das eigentliche Menü öffnet, wenn man mit dem Mauszeiger darauf zufährt. Solche Oberbegriffe müssen, jedes für sich, in einem Record mit dem Namen Menu definiert werden:

```
Menu=RECORD
  nextMenu : MenuPtr    ; Zeiger auf den nächsten Oberbegriff
  leftEdge : INTEGER    ; x-Koordinate der linken, oberen Ecke des Oberbegriffs
                          (absolut!)
  topEdge  : INTEGER    ; y-Koordinate der linken, oberen Ecke des Oberbegriffs
                          (absolut!)
```

```
  width    : INTEGER    ; Breite des Oberbegriffs in Bildpunkten
  height   : INTEGER    ; Höhe des Oberbegriffs in Bildpunkten
  flags    : BITSET     ; Flags des Menüs
  menuName : ADDRESS    ; Adresse des Strings mit dem Text des Oberbegriffs
  firstItem: MenuItemPtr; Zeiger auf den ersten Menüpunkt DIESES Menüs
  jazzX    : INTEGER    ; für internen Gebrauch
  jazzY    : INTEGER    ; für internen Gebrauch
  beatX    : INTEGER    ; für internen Gebrauch
  beatY    : INTEGER    ; für internen Gebrauch
END;
```

Zu flags: Steht hier {0}, so ist das Menü anwählbar, schreiben Sie aber »leere Menge« hinein, so wird das entsprechende Menü punktiert dargestellt (kennen wir von den Gadgets) und ist folglich nicht anwählbar. Der noch verbleibende Zustand (beim Benchmark-CompilerMiDrawn genannt und Typ MenuFlagSet) wird von Intuition verwaltet und heißt, daß die Menüpunkte dieses Oberbegriffs augenblicklich zu sehen sind (beim M2-Compiler ist dieses Flag eine Konstante und heißt »iDrawnm«).

3.5.2 Items

Zu einem richtigen Menü gehören natürlich auch anständige Menüpunkte (sonst hätten sie ja keine Existenzberechtigung). Und der Programmierung dieser Menüpunkte (bzw. »Items«, wie sie häufig genannt werden) wollen wir uns jetzt zuwenden. Auch hier werden die Informationen wieder in einem Record zusammengetragen, der MenuItem heißt:

```
MenuItem=RECORD
  nextItem      : MenuItemPtr    ; Zeiger auf das nächste Item DIESES Menüs
  leftEdge      : INTEGER        ; x-Koordinate der linken, oberen Ecke des
                                   Items (relativ zum Menü)
  topEdge       : INTEGER        ; y-Koordinate der linken, oberen Ecke des
                                   Items (relativ zum Menü)
  width         : INTEGER        ; Breite des Items
  height        : INTEGER        ; Höhe des Items
  flags         : MenuItemFlagSet; Item-Flags
  mutualExclude: LONGSET         ; Definition von sich ausschließenden Menüpunkten
  itemFill      : ADDRESS        ; Art der Darstellung des Items
  selectFill    : ADDRESS        ; Art der Darstellung beim Anwählen
  command       : CHAR           ; äquivalente Taste (Shortcut)
  subItem       : MenuItemPtr    ; Zeiger auf den ersten Untermenüpunkt
  nextSelect    : CARDINAL       ; wird von Intuition belegt, wenn der Anwender
                                   ein Item angewählt hat
END;
```

flags In diesem Feld können Sie Ihr Item näher bestimmen. Als Angabe zulässig sind alle Werte der Menge MenuItemFlagSet:

checkIt Durch Setzen dieses Flags bewirken Sie, daß der entsprechende Menüpunkt bei der Anwahl »abgehakt« (also mit einem Haken versehen) wird. Sollte ein bereits abgehakter Menüpunkt nochmals angewählt werden, verschwindet der Haken. Im Feld checkMark des Window-Records können Sie, wenn Sie sich erinnern, Ihren eigenen Haken definieren.

itemText Statt eines Bildchens (Image) wird ein Text im Item dargestellt, was der Normalfall ist.

commSeq Setzen Sie dieses Flag, akzeptiert Intuition anstelle eines Mausklicks auch eine Tastenkombination (Shortcut) für die Anwahl eines Items. Die Taste, die in Verbindung mit <Amiga rechts> gedrückt werden muß, um den Menüpunkt anzuwählen, muß im Feld definiert werden.

menuToggle Sie müssen dieses Flag setzen, wenn Sie erlauben, daß der Benutzer durch wiederholtes Anklicken ein abgehaktes CHECKIT-Item wieder »normalisieren« kann (der Haken verschwindet).

itemEnabled Wird dieses Flag gesetzt, so ist das Item eingeschaltet. Wenn Sie aber ein ausgeschaltetes Item möchten, so lassen Sie dieses Flag ungesetzt. Ein ausgeschaltetes Item sieht aus wie ein deaktivierter Oberbegriff oder ein deaktiviertes Gadget – gepunktet nämlich.

mif5 Dieses Flag ist für den internen Gebrauch.

highComp Steht der Mauszeiger über einem Item, welches dieses Flag gesetzt hat, so wird ersteres invertiert (komplementiert) dargestellt.

highBox Eine sinnvolle Alternative zu HIGHCOMP – ein Item, über dem der Mauszeiger steht, wird umrahmt.

checked Ist dieses Flag gesetzt, so wird das entsprechende Item, das zusätzlich mit CHECKIT gekennzeichnet sein muß, gleich zu Beginn abgehakt (also angewählt) dargestellt. Wird ein anfangs »unabgehaktes« CHECKIT-Item abgehakt, setzt Intuition deses Flag automatisch.

mif9–mif11 Nur für den internen Gebrauch!

isDrawn Intuition setzt dieses Flag, wenn das betreffende Item gerade zu sehen ist.

highItem — Intuition setzt dieses Flag, wenn der Mauszeiger über dem entsprechenden Menüpunkt steht.

menuToggled — Für Mehrfachanwahl eines Menüpunkts.

highNone (bei M2Amiga Konstante!) — Ein Item wird, wenn der Mauszeiger über ihm steht, nicht verändert.

highImage — Steht der Mauszeiger über einem Item, so wird das Image (oder der Text), auf den selectFill zeigt, dargestellt. Mit diesem Flag sind übrigens ganz interessante Effekte möglich: Sie können z.B. ein Item, das die Funktion auslöst, das Programm zu beenden, so präparieren, daß sich der Text meinetwegen von »ENDE« in »SICHER??« verwandelt. Da die Definition dieses Flags bei meinem M2Amiga (Version 3.11) fehlt, sei gesagt, daß HIGHIMAGE automatisch gesetzt ist, wenn kein anderes »high«-Flag (also weder HIGHCOMP noch HIGHBOX noch HIGHNONE) gesetzt ist.

mutualExclude — Mit diesem Feld kann man bestimmen, welche (CHECKIT-) Items sich gegenseitig ausschließen (wie bei den Gadgets, nur daß es hier funktioniert). mutualExclude ist nicht gerade einfach zu programmieren, so daß eine genaue Erklärung und ein Beispiel folgt: Jedes Bit im LONGSET-Wert repräsentiert ein Item eines Menüs (mit 32 Bits werden Sie wohl auskommen). Bit 0 bezieht sich auf das erste Item, Bit 1 auf das zweite usw. Wird ein Bit auf 1 gesetzt, so darf das Item, dessen Bit gesetzt ist, nicht zusammen mit diesem Item gewählt sein. Ein nicht gesetztes Bit heißt somit, daß das Item mit diesem zusammen eingeschaltet werden darf. Beispiel mit drei Items:

Item 1: LONGSET{2}
Item 2: LONGSET{2}
Item 3: LONGSET{0,1}

Das bedeutet, daß die Items 1 und 2 gleichzeitig eingeschaltet sein dürfen. Doch darf Item 3 nur eingeschaltet werden, wenn 1 und 2 ausgeschaltet sind.

itemFill — Dieses Feld muß die Adresse auf das Füllobjekt des Items beinhalten. Haben Sie das Flag ITEMTEXT gesetzt, so muß hier die Adresse der entsprechenden IntuiText-Struktur stehen, haben Sie ein Image gewählt, dementsprechend ein Zeiger darauf.

selectItem Wünschen Sie eine Itemveränderung, wenn der Mauszeiger darüber steht, so muß hier ein Zeiger auf das neue Aussehen des Items stehen. Haben Sie bei itemFill den Zeiger auf einen Text eingetragen, muß auch hier der eines Textes stehen. Steht in itemFill ein Image-Zeiger, muß natürlich auch hier einer stehen.

subItem Hier wird einfach der Zeiger auf ein Untermenü (falls Sie ein solches wünschen) eingetragen. Das Untermenü erscheint, wenn sich der Mauszeiger über dem zugehörigen Item befindet.

3.5.3 Menüauswertung

Nun wollen wir uns an die Menüauswertung machen, was aber gar nicht mal so einfach ist, gerade beim M2Amiga-Compiler. Denn da haben die Programmierer anscheinend dummerweise (wenn ich härter werden darf: blöderweise) vergessen, die Makros MENUNUM, ITEMNUM und SUBNUM der C-Compiler zu implementieren, was beim Benchmark-Compiler nicht der Fall ist. So müssen wir sehen, daß wir uns eigene Auswertungsroutinen schreiben.

Zuerst sollte man sich klarmachen, in welcher Form die Menünummern ankommen:

```
SSSSS III  III MMMMM (Doppelbyte = CARDINAL)
```

Fangen wir mit den Untermenüs (S) an: Sie belegen die ersten 5 Bits des ersten Bytes. Um an sie heranzukommen, muß man den ganzen Wert einfach um 11 Bits nach rechts schieben (SHIFTen), so daß die ersten fünf Bits dann freiliegen und den Untermenüpunkt angeben.

Bei den Items (I) wird es schon schwieriger: um die Menünummer loszuwerden, schieben wir den Wert 5 Bits nach rechts. Da die Menübits dann unglücklicherweise links zusammen mit den Untermenüs wieder auftauchen, schieben wir den Term 10 Bits nach links und anschließen wieder 10 nach rechts. So kommen wir auch an die Items.

Die Auswertung des Menüpunktes (M) ist dann recht einfach: um Items und Subitems loszuwerden, schieben wir die Zahl 11 Bits nach links und anschließen nach rechts.

Da der oben geschilderte Vorgang recht umständlich zu handhaben ist, halte ich es für das beste, drei Funktionsprozeduren, die die ganze Schieberei übernehmen, zu schreiben:

```
PROCEDURE MenuNum(Num: CARDINAL): CARDINAL;
  BEGIN
    RETURN(SHIFT(SHIFT(Num,11),-11);
  END MenuNum;
```

```
PROCEDURE ItemNum(Num: CARDINAL): CARDINAL;
  BEGIN
    RETURN(SHIFT(SHIFT(SHIFT(Num,-5),10),-10));
  END ItemNum;

PROCEDURE SubNum(Num: CARDINAL): CARDINAL;
  BEGIN
    RETURN(SHIFT(Num,-11));
  END SubNum;
```

Somit wäre das erste Problem gelöst. Doch wie kommt man nun an die kodierte Zahl und was macht man, wenn man sie ausgewertet hat? Ganz einfach, man setzt zunächst das IDCMP-Flag MENUPICK in der Window-Struktur. Dann initialisiert man das Menü mit SETMENUSTRIP:

set := SetMenuStrip (MeinWindow , ErstesMenü);

set	BOOLEAN-Wert, der anzeigt, ob das Menü erfolgreich initialisiert wurde (TRUE) oder ob ein Fehler aufgetreten ist (FALSE).
MeinWindow	WindowPtr auf das Fenster, für das das Menü gelten soll.
ErstesMenü	MenuPtr auf das erste Menü der Liste.

Nun kommt wieder unsere flexible IDCMP-Auswertung zum Zuge: Ist das Flag MENUPICK im Feld class, so wurde ein Menüpunkt ausgewählt. Im Feld code steht die kodierte Nummer des Menüpunktes (s.o.), die wir dann mit einer CASE-Anweisung auswerten können.

3.5.4 Menü-Befehle

Auch für die Menüs gibt es einige Befehle, die hier genannt werden sollen: Am Programmende sollte man ein mit SETMENUSTRIP initialisiertes Menü auch wieder löschen. Dazu gibt es den CLEARMENUSTRIP-Befehl:

ClearMenuStrip (MeinWindow);

MeinWindow	WindowPtr auf das Fenster, dessen Menüleiste gelöscht werden soll.

Aber auch um Menüoberpunkte nachträglich ein- und auszuschalten, kann man sich zweier Intuition-Befehle bedienen, sie heißen trefflicherweise ONMENU und OFF-MENU:

OnMenu (MeinWindow , Menüpunkt);

MeinWindow	WindowPtr auf das Fenster, dessen Menüleiste manipuliert werden soll.
Menüpunkt	CARDINAL-Wert des Menüpunktes, der eingeschaltet werden soll.

OffMenu (MeinWindow , Menüpunkt);

MeinWindow	WindowPtr auf das Fenster, dessen Menüleiste manipuliert werden soll.
Menüpunkt	CARDINAL-Wert des Menüpunktes, der ausgeschaltet werden soll.

Wollen Sie aus irgendeinem Grund die Adresse eines Items heraussuchen, so sollten Sie den Befehl ITEMADDRESS benutzen:

Adresse := ItemAddress (MeinMenü , Itemnummer);

MeinMenü	MenuPtr auf das Menü, aus dessen Liste ein Item herausgesucht werden soll.
Itemnummer	CARDINAL-Nummer des Items, dessen Adresse Sie haben möchten.
Adresse	MenuItemPtr auf das gefundene Item
Achtung:	Dieser Befehl funktioniert nicht bei Untermenüs!

Die Menüprogrammierung dürfte nun keine Probleme mehr bereiten, und wir wollen ein neues Beispielprogramm studieren:

```
MODULE MenuDemo;

FROM SYSTEM    IMPORT ADDRESS,ADR,LONGSET,SHIFT;
FROM Intuition IMPORT NewWindow,WindowPtr,IDCMPFlags,IDCMPFlagSet,
                      WindowFlags,WindowFlagSet,OpenWindow,CloseWindow,
                      IntuiText,Menu,MenuItem,GadgetFlags,MenuItemFlagSet,
                      ScreenFlags,ScreenFlagSet,SetMenuStrip,ClearMenuStrip,
                      IntuiMessagePtr,MenuItemFlags;
FROM Graphics  IMPORT jam1;
FROM Exec      IMPORT GetMsg,ReplyMsg;
```

```
VAR WindowDaten: NewWindow;
    MeinWindow : WindowPtr;
    MeinText   : ARRAY[Ø..3] OF IntuiText;
    MeinMenu   : ARRAY[Ø..1] OF Menu;
    MeinItem   : ARRAY[Ø..3] OF MenuItem;
    ok         : BOOLEAN;
    IntuiMsg   : IntuiMessagePtr;
    class      : IDCMPFlagSet;
    code       : CARDINAL;

PROCEDURE ItemNum(Num: CARDINAL): CARDINAL;
  BEGIN
    RETURN(SHIFT(SHIFT(SHIFT(Num,-5),1Ø),-1Ø));
  END ItemNum;

PROCEDURE MenuNum(Num: CARDINAL): CARDINAL;
  BEGIN
    RETURN(SHIFT(SHIFT(Num,11),-11));
  END MenuNum;

BEGIN
  WITH WindowDaten DO (* Fenster initialisieren — kalter Kaffee *)
    leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256;
    detailPen:=Ø; blockPen:=1; idcmpFlags:=IDCMPFlagSet{menuPick};
    flags:=WindowFlagSet{windowSizing,windowDrag,windowDepth,activate};
    firstGadget:=NIL; checkMark:=NIL; title:=ADR("Menü-Window");
    bitMap:=NIL; type:=ScreenFlagSet{wbenchScreen}; minWidth:=4ØØ;
    maxWidth:=64Ø; minHeight:=2Ø; maxHeight:=256;
  END;

  WITH MeinMenu[Ø] DO
    nextMenu:=ADR(MeinMenu[1]); leftEdge:=1Ø; topEdge:=Ø; width:=15Ø;
    height:=Ø; flags:={Ø}; menuName:=ADR("ErstesMenü");
    firstItem:=ADR(MeinItem[Ø]);
  END;

  WITH MeinMenu[1] DO
    nextMenu:=NIL; leftEdge:=17Ø; topEdge:=Ø; width:=15Ø; height:=9;
    flags:={Ø}; menuName:=ADR("ZweitesMenü"); firstItem:=ADR(MeinItem[1]);
  END;

  WITH MeinItem[Ø] DO
    nextItem:=NIL; leftEdge:=-5; topEdge:=Ø; width:=15Ø; height:=11;
    flags:=MenuItemFlagSet{highComp,itemText,itemEnabled,commSeq};
    mutualExclude:=LONGSET{}; itemFill:=ADR(MeinText[Ø]); selectFill:=NIL;
    subItem:=NIL; command:="Q";
  END;
```

```
WITH MeinItem[1] DO
  nextItem:=ADR(MeinItem[2]); leftEdge:=-5; topEdge:=0; width:=150;
  height:=11; subItem:=NIL;
  flags:=MenuItemFlagSet{highComp,itemText,itemEnabled,checkIt,menuToggle};
  mutualExclude:=LONGSET{2}; itemFill:=ADR(MeinText[1]); selectFill:=NIL;
END;

WITH MeinItem[2] DO
  nextItem:=ADR(MeinItem[3]); leftEdge:=-5; topEdge:=12; width:=150;
  height:=11; subItem:=NIL;
  flags:=MenuItemFlagSet{highComp,itemText,itemEnabled,checkIt,menuToggle};
  mutualExclude:=LONGSET{2}; itemFill:=ADR(MeinText[2]); selectFill:=NIL;
END;

WITH MeinItem[3] DO
  nextItem:=NIL; leftEdge:=-5; topEdge:=24; width:=150; height:=11;
  flags:=MenuItemFlagSet{highComp,itemText,itemEnabled,checkIt,menuToggle};
  mutualExclude:=LONGSET{0,1}; itemFill:=ADR(MeinText[3]); selectFill:=NIL;
  subItem:=NIL;
END;

WITH MeinText[0] DO
  nextText:=NIL; frontPen:=0; backPen:=1; drawMode:=jam1;
  leftEdge:=0; topEdge:=0; iTextFont:=NIL; iText:=ADR("ENDE");
END;

WITH MeinText[1] DO
  nextText:=NIL; frontPen:=0; backPen:=1; drawMode:=jam1;
  leftEdge:=20; topEdge:=0; iTextFont:=NIL; iText:=ADR("Check 1");
END;

WITH MeinText[2] DO
  nextText:=NIL; frontPen:=0; backPen:=1; drawMode:=jam1;
  leftEdge:=20; topEdge:=0; iTextFont:=NIL; iText:=ADR("Check 2");
END;

WITH MeinText[3] DO
  nextText:=NIL; frontPen:=0; backPen:=1; drawMode:=jam1;
  leftEdge:=20; topEdge:=0; iTextFont:=NIL; iText:=ADR("Check 3");
END;

MeinWindow:=OpenWindow(WindowDaten);
ok:=SetMenuStrip(MeinWindow,ADR(MeinMenu[0]));

IF (MeinWindow#NIL) THEN
  LOOP
    IntuiMsg:=GetMsg(MeinWindow^.userPort);
    WHILE IntuiMsg#NIL DO
```

```
        class:=IntuiMsg^.class;
        code :=IntuiMsg^.code;
        ReplyMsg(IntuiMsg);
        IF (menuPick IN class) THEN
          IF MenuNum(code)=Ø THEN
            IF ItemNum(code)=Ø THEN EXIT; END;
          END;
        END;
        IntuiMsg:=GetMsg(MeinWindow^.userPort);
      END;
    END;
    ClearMenuStrip(MeinWindow);
    CloseWindow(MeinWindow);
  END;
END MenuDemo.
```

3.6 Requester

Requester sind Ansammlungen von Gadgets. Auch der Compiler-Driver des vorletzten Kapitels hätte ein Requester sein können, nur wäre er dann umständlicher zu programmieren gewesen. Requester kann man in drei große Gruppen unterteilen:

1. Auto-Requester (auch System-Requester). Das sind die vom System erstellten und gehandhabten JA-/NEIN-Auswahlfelder. Ein bekanntes Beispiel ist der »Please insert volume ... in any drive«-Requester, er läßt nur zwei Antworten zu. Meist sind diese JA/NEIN oder WEITER/ABBRUCH. Auto-Requester sind am einfachsten zu programmieren und zu handhaben.

2. Custom-Requester. Damit bezeichnet man »selbstgestrickte« Requester, die der Programmierer selbst definieren und steuern muß. Zu dieser Gruppe kann man auch die File-Requester, wie sie viele professionelle Programme beim Laden und Speichern von Dateien anzeigen, zählen.

3. Double-Menu-Requester. Dies ist wohl die sonderbarste Gruppe der drei: Double-Menu-Requester erscheinen nur bei doppeltem Klick der rechten Maustaste. Auch sie sind selbstdefinierbar, wie die Custom-Requester.

3.6.1 Auto-Requester

Fangen wir mit Auto-Requestern an. Da sie recht einfach aufgebaut sind und nur eine von zwei Antworten zulassen, können diese Requester mit einem einfachen Befehl dargestellt werden:

ok := AutoRequest (MeinWindow , Überschrift , JaText , NeinText , JaFlags , NeinFlags , Breite , Höhe);

MeinWindow	WindowPtr auf das Fenster, in dem der Requester dargestellt werden soll. Wird hier NIL angegeben, so bildet Intuition selbst ein Fenster.
Überschrift	IntuiTextPtr auf den IntuiText, der die Überschrift des Requesters (z.B. »Wollen Sie das Programm wirklich beenden?«) beinhaltet.
JaText	IntuiTextPtr auf die positive Antwort (Text im linken Gadget, z.B. »JA« oder »WEITER«).
NeinText	IntuiTextPtr auf die negative Antwort (Text im rechten Gadget, z.B. »NEIN« oder »ABBRUCH«).
JaFlags	IDCMPFlagSet für die Gadget-Abfrage des positiven Textes. Da Intuition jedoch die Verwaltung der Gadgets übernimmt, können Sie hier ruhig »leere Menge« eintragen.
NeinFlags	IDCMPFlagSet für die Gadget-Abfrage des negativen Textes. Da Intuition aber die Abfrage der Gadgets übernimmt, können Sie hier ruhig »leere Menge« eintragen.
Breite	INTEGER-Wert, der die Breite des Requesters in Bildpunkten angibt.
Höhe	INTEGER-Wert, der die Höhe des Requesters in Bildpunkten angibt.
ok	BOOLEAN-Wert des Ergebnisses, ist entweder TRUE, falls das »JA«-Gadget oder negativ, wenn das »NEIN«-Gadget angewählt wurde.

3.6.2 Custom-Requester

Oftmals genügen die von Intuition bereitgestellten Auto-Requester nicht. Dafür gibt es die Möglichkeit, sich selbst Requester zusammenzubauen, die Custom-Requester, mit denen von der simplen JA-/NEIN-Abfrage bis hin zum komfortablen Dateiauswahlfeld alles möglich ist. Da diese Art von Requester recht viele Parameter braucht, muß man einen Record mit dem Namen Requester initialisieren:

```
Requester=RECORD
  olderRequest: RequesterPtr          ; Zeiger auf vorher bearbeiteten Requester
  leftEdge    : INTEGER               ; x-Koordinate der linken, oberen Ecke
  topEdge     : INTEGER               ; y-Koordinate der linken, oberen Ecke
  width       : INTEGER               ; Breite des Requesters
  height      : INTEGER               ; Höhe des Requesters
  relLeft     : INTEGER               ; x-Koordinate relativ zum Mauszeiger
  relTop      : INTEGER               ; y-Koordinate relativ zum Mauszeiger
  reqGadget   : GadgetPtr             ; Zeiger auf das erste Gadget
  reqBorder   : BorderPtr             ; Zeiger auf den Rahmen des Requester oder
                                        NIL bei keinem Rahmen
  reqText     : IntuiTextPtr          ; Zeiger auf den ersten Text im Requester
  flags       : RequesterFlagSet      ; Requester-Flags
  backFill    : Byte                  ; Farbe des Requester-Hintergrundes
  reqLayer    : LayerPtr              ; Zeiger auf das Layer des Requesters,
                                        meist jedoch NIL
  reqPad1     : ARRAY[Ø..31] OF BYTE ; für die interne Benutzung reserviert
  imageBMap   : BitMapPtr             ; für eigene BitMap, meist jedoch NIL
  rWindow     : WindowPtr             ; System-Variable (nichts für uns)
  reqPad2     : ARRAY[Ø..35] OF BYTE ; für die interne Benutzung reserviert
END;
```

olderRequest — Hier steht ein Zeiger auf den Requester, den Intuition vorher noch zu bearbeiten hat (wird von Intuition verwaltet).

relLeft — Möchte man einen Requester relativ zur aktuellen Mausposition erscheinen lassen, so gibt man hier den x-Abstand an. In einem solchen Fall muß das POINTREL-Flag gesetzt sein!

relTop — Soll ein Requester relativ zur aktuellen Mausposition erscheinen, so muß hier der y-Abstand angegeben werden. Das POINTREL-Flag muß gesetzt sein!

reqGadget — In diesem Feld muß ein Zeiger auf das erste Gadget des Requesters stehen. Achtung: mindestens ein Gadget muß das Activation-Flag ENDGADGET gesetzt haben, damit der Requester geschlossen wird!

reqText — Da man in manchen Requestern NUR Text darstellen will (»Info«-Requester z.B., in denen der Programmautor und seine Adresse stehen), muß in diesem Feld ein Zeiger auf den ersten Text stehen. Weil jedoch eine Zeile meist nicht ausreicht, können (oder müssen) die verschiedenen IntuiText-Records über das Feld nextText miteinander verbunden sein.

flags In diesem Feld sind alle Werte der Menge RequesterFlagSet zulässig:

pointRel Dieses Flag muß gesetzt sein, um den Requester relativ zur Mausposition erscheinen zu lassen. Die Koordinaten für die relative Position müssen in den Feldern relLeft und relTop angegeben werden.

preDrawn Wenn Sie eine eigene Bitmap für den Requester verwenden möchten, so müssen Sie dieses Flag setzen.

noisyReq Wenn Sie die Requestereingaben nicht filtern wollen, so müssen Sie dieses Flag setzen.

Die folgenden Flags werden von Intuition gesetzt:

reqOffWindow Dieses Flag ist gesetzt, wenn der Requester, nicht aber sein Window aktiviert ist.

reqActive Ist dieses Flag gesetzt, so ist der Requester augenblicklich aktiv.

sysRequest Ist der Requester ein Systemrequester, so ist dieses Flag gesetzt.

deferRefresh Erscheint der Requester, wird eine Refresh-Meldung gestoppt.

Die restlichen Flags (rf3–rf11) sind für das System reserviert.

Haben Sie diesen Record ausreichend ausgefüllt (was Ihnen wohl kein Problem mehr bereiten dürfte), so können Sie Ihren Requester mit dem Befehl REQUEST »herbeizaubern«:

ok := Request (MeinRequester , MeinWindow);

MeinRequester	RequesterPtr auf die Struktur des Requesters.
MeinWindow	WindowPtr auf das Fenster, in dem der Requester erscheinen soll.
ok	BOOLEAN-Rückgabewert; ist TRUE, wenn der Requester erfolgreich dargestellt werden konnte oder FALSE, wenn irgendein Fehler auftrat.

Es versteht sich wohl von selbst, daß Sie die Abfrage der Requester-Gadgets selbst übernehmen müssen. Noch ein Wort zu den Requester-Gadgets: Nebst ihrer Typen-Kennzeichnung wie boolGadget müssen Sie hier noch die Konstante reqGadget hinzuaddieren (oder ORen). Als IDCMPFlag sollten Sie unbedingt REQCLEAR setzen, damit Sie Nachricht erhalten, ob der Requester geschlossen wurde, was Intuition automatisch beim Anklicken eines mit ENDGADGET gekennzeichneten Gadgets macht.

3.6.3 Double-Menu-Requester

Double-Menu-Requester unterscheiden sich von den Custom-Requestern nur in der Tatsache, daß sie durch zweimaliges Drücken der rechten Maustaste erscheinen. Auch für sie muß der Requester-Record ausgefüllt werden. Sie sind mit dem Befehl SETDM-REQUEST zu initialisieren:

ok := SetDMRequest (MeinWindow , MeinRequester);

MeinWindow	WindowPtr auf das Fenster, in dem der Requester nach zweimaligem Drücken der rechten Maustaste erscheinen soll.
MeinRequester	RequesterPtr auf die Requester-Struktur des DM-Requesters.
ok	BOOLEAN-Rückgabewert, ist TRUE, wenn der Requester erfolgreich initialisiert werden konnte oder FALSE, wenn ein Fehler auftrat (es ist zum Beispiel schon ein DM-Requester initialisiert).

Wollen Sie einen DM-Requester löschen, so können Sie dazu den Befehl CLEARDM-REQUEST verwenden:

ok := ClearDMRequest (MeinWindow);

MeinWindow	WindowPtr auf das Fenster, dessen DM-Requester gelöscht werden soll.
ok	BOOLEAN-Rückgabewert, ist TRUE, wenn der Requester entfernt werden konnte oder FALSE, wenn das Gegenteil der Fall ist.

3.6.4 Die Requester-Befehle

Neben den vier Requester-Befehlen, die wir bereits kennengelernt haben, gibt es drei weitere, die den Umgang mit Requestern vereinfachen. Um einen (Custom-)Requester aus der Liste eines Fensters zu löschen, muß man lediglich ENDREQUEST angeben:

EndRequest (MeinRequester , MeinWindow);

MeinRequester	RequesterPtr auf den Record des zu entfernenden Requesters.
MeinWindow	WindowPtr auf das Fenster, aus dessen Liste der Requester gelöscht werden soll.

Möchten Sie einen System-Requester initialisieren und diesen selbst »managen«, sollten Sie sich den Befehl BUILDSYSREQUEST einmal ansehen (diese Funktion wird auch von AUTOREQUEST aufgerufen):

Window = BuildSysRequest (MeinWindow , Überschrift , JaText ,
NeinText , Flags , Breite , Höhe);

MeinWindow	WindowPtr auf das Fenster, in dem der Requester erscheinen soll. Wird hier NIL angegeben, so erzeugt Intuition ein Window.
Überschrift	IntuiTextPtr auf den Titeltext des Requesters (z.B. »Möchten Sie das Programm wirklich beenden« aber auch »Bitte legen Sie die Diskette ... in irgendein Laufwerk!«).
JaText	IntuiTextPtr des Gadgets für die positive Antwort (dieses Gadget befindet sich links).
NeinText	IntuiTextPtr des Gadgets für die negative Antwort (dieses Gadget befindet sich rechts).
Flags	IDCMPFlagSet-Wert des Fensters.
Breite	Breite des Requesters in Bildpunkten.
Höhe	Höhe des Requesters in Bildpunkten.
Window	WindowPtr auf das Fenster, zu dem der Requester gehört.

Alle mit BUILDSYSREQUEST (und daher auch mit AUTOREQUEST) generierten Requester-Gadgets haben folgende Flags und Typenkennzeichnungen:

BOOLGADGET	klar, die Gadgets sind einfache Schalter.
RELVERIFY	das Programm erhält sofort Nachricht, wenn ein Gadget betätigt wurde.
REQGADGET	zeichnet die Gadgets als Requester-Gadgets aus.
TOGGLESELECT	das Gadget kann nur die beiden Zustände »angewählt« und »nicht angewählt« haben.

Um der System-Requester wieder ledig zu werden, gibt es den Befehl FREESYSREQUEST, der alle mit BUILDSYSREQUEST generierten Requester aus der Liste eines Fensters löscht:

FreeSysRequest (MeinWindow);

MeinWindow	WindowPtr auf das Fenster, aus dessen Liste alle BUILDSYS-REQUEST-Requester gelöscht werden sollen.

Damit hätten wir auch das Kapitel über Requester abgeschlossen. Zur Verdeutlichung soll nun das obligatorische Beispielprogramm folgen:

```
MODULE RequesterDemo;

FROM SYSTEM    IMPORT ADR,LONGSET;
FROM Intuition IMPORT NewWindow,WindowPtr,OpenWindow,CloseWindow,Requester,
                      RequesterFlags,Gadget,GadgetFlags,ActivationFlags,
                      RequesterFlagSet,GadgetFlagSet,ActivationFlagSet,
                      IntuiText,Border,Request,IDCMPFlags,WindowFlagSet,
                      IDCMPFlagSet,ScreenFlags,ScreenFlagSet,WindowFlags;
FROM Graphics  IMPORT jam1;
FROM Exec      IMPORT SetSignal;

VAR WindowDaten  : NewWindow;
    MeinWindow   : WindowPtr;
    MeinRequester: Requester;
    xyFeld       : ARRAY[Ø..9] OF INTEGER;
    MeinBorder   : Border;
    MeinGadget   : Gadget;
    MeinText     : ARRAY[Ø..4] OF IntuiText;
    GadgetText   : IntuiText;
    ok   : BOOLEAN;

BEGIN
  WITH WindowDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256; detailPen:=Ø; blockPen:=1;
    idcmpFlags:=IDCMPFlagSet{reqClear}; flags:=WindowFlagSet{activate,
    windowDepth,windowDrag}; firstGadget:=NIL; checkMark:=NIL;
    title:=ADR("Requester-Demo"); screen:=NIL; bitMap:=NIL; minWidth:=Ø;
    maxWidth:=Ø; minHeight:=Ø; maxHeight:=Ø; type:=ScreenFlagSet{wbenchScreen};
  END;

  WITH MeinText[Ø] DO
    nextText:=ADR(MeinText[1]); frontPen:=2; backPen:=1; drawMode:=jam1;
    leftEdge:=13; topEdge:=5; iTextFont:=NIL;
    iText:=ADR("Dieses Programm wurde geschrieben von:");
  END;
```

```
WITH MeinText[1] DO
  nextText:=ADR(MeinText[2]); frontPen:=2; backPen:=1; drawMode:=jam1;
  leftEdge:=13; topEdge:=21; iTextFont:=NIL;
  iText:=ADR("Holger Gzella");
END;

WITH MeinText[2] DO
  nextText:=ADR(MeinText[3]); frontPen:=2; backPen:=1; drawMode:=jam1;
  leftEdge:=13; topEdge:=31; iTextFont:=NIL;
  iText:=ADR("Freischöffenweg 12");
END;

WITH MeinText[3] DO
  nextText:=ADR(MeinText[4]); frontPen:=2; backPen:=1; drawMode:=jam1;
  leftEdge:=13; topEdge:=41; iTextFont:=NIL;
  iText:=ADR("4600 Dortmund 16");
END;

WITH MeinText[4] DO
  nextText:=NIL; frontPen:=2; backPen:=1; drawMode:=jam1;
  leftEdge:=13; topEdge:=51; iTextFont:=NIL;
  iText:=ADR("Tel.: (0231) 85 16 02");
END;

WITH GadgetText DO
  nextText:=NIL; frontPen:=3; backPen:=1; drawMode:=jam1;
  leftEdge:=1; topEdge:=2; iTextFont:=NIL; iText:=ADR("OK!");
END;

WITH MeinBorder DO
  leftEdge:=-1; topEdge:=-1; frontPen:=3; backPen:=2; drawMode:=jam1;
  count:=5; xy:=ADR(xyFeld); nextBorder:=NIL;
END;

WITH MeinGadget DO
  nextGadget:=NIL; leftEdge:=144; topEdge:=63; width:=30; height:=10;
  flags:=GadgetFlagSet{}; activation:=ActivationFlagSet{relVerify,
  gadgImmediate,endGadget}; gadgetType:=1001H (* boolGadget + reqGadget *);
  gadgetRender:=ADR(MeinBorder); selectRender:=NIL; gadgetID:=1;
  gadgetText:=ADR(GadgetText); specialInfo:=NIL; userData:=NIL;
END;

xyFeld[0]:=0; xyFeld[1]:=0; xyFeld[2]:=31; xyFeld[3]:=0;
xyFeld[4]:=31; xyFeld[5]:=11; xyFeld[6]:=0; xyFeld[7]:=11;
xyFeld[8]:=0; xyFeld[9]:=0;
```

```
  WITH MeinRequester DO
    olderRequest:=NIL; leftEdge:=0; topEdge:=0; width:=330; height:=76;
    relLeft:=0; relTop:=0; reqGadget:=ADR(MeinGadget); reqBorder:=NIL;
    reqText:=ADR(MeinText[0]); flags:=RequesterFlagSet{}; backFill:=1;
    reqLayer:=NIL; imageBMap:=NIL; rWindow:=NIL;
  END;

  MeinWindow:=OpenWindow(WindowDaten);
  ok:=Request(ADR(MeinRequester),MeinWindow);
  WHILE NOT (MeinWindow^.userPort^.sigBit IN SetSignal(LONGSET{},LONGSET{})) DO
  END;
  CloseWindow(MeinWindow);
END RequesterDemo.
```

Um einen Double-Menu Requester darzustellen, müssen Sie die Zeile mit dem Befehl

```
ok:=Request(ADR(MeinRequester),MeinWindow);
```

in

```
ok:=SetDMRequest(MeinWindow,ADR(MeinRequester));
```

ändern und den Befehl SETDMREQUEST in die Intuition-Importliste anfügen.

3.7 Alerts

Bei diesem Wort schaudert jeder Programmierer – Alerts, auch Guru-Meditationen genannt. Für eine genaue Übersicht der letztgenannten siehe auch Anhang A. Man kann aber auch eigene Alerts mit komplett eigenen Meldungen (»Der Guru-Master grüßt alle Fans von Richard Wagner«) produzieren. Wie, das wird nun gezeigt.

Zuerst müssen wir uns klarmachen, wie eine Alertmeldung aufgebaut ist:

```
x  y              T E X T              Z1 Z2
```

Der ganze Term ist ein String mit nur sovielen Zeichen, wie unbedingt notwendig. »x« ist ein 16-Bit-Wert (für uns CARDINAL), der die x-Koordinate des Textes angibt, »y« ist ein 8-Bit-Wert (am besten CHAR(y)) für die y-Koordinate, »TEXT« ist die eigentliche Meldung, »Z1« ist das Endkennzeichen des Textes, muß CHAR(0) sein und »Z2« ist das Endkennzeichen der Meldung. Hier steht entweder CHAR(0), wenn die Meldung zuende ist oder ein Wert größer als 0, ich nehme immer CHAR(1)), w(enn eine weitere Meldung folgen soll. Diese ist natürlich genauso wie die erste aufgebaut.

Modula-2-Programmierer sollten hier einen Record verwenden:

```
Alert=RECORD
  x1    : CARDINAL              ; x-Koordinate
  Zeile1: ARRAY[Ø..59] OF CHAR ; 1.Zeile, nur soviele Zeichen, wie man
                                 unbedingt braucht
  x2    : CARDINAL              ; x-Koordinate für 2.Zeile
  Zeile2: ARRAY[Ø..59] OF CHAR ; 2.Zeile
  ...                           ; evt. noch 3.-x. Zeile + x-Werte
END;
```

Meistens reichen jedoch zwei Zeilen aus. Und nun müssen wir uns Gedanken über den Typ des Alerts machen:

1. recoveryAlert (00000000H): Wenn der Alert erscheint, wird der übrige Bildschirm nach unten geschoben. Der Alert hat keine Folgewirkungen und ist nur dazu da, den Benutzer auf etwas aufmerksam zu machen.

2. deadEndAlert (80000000H): Nach der Darstellung dieses Alerts wird ein Reset ausgeführt. Das System benutzt diesen Typ bei solch gravierenden Fehlern, daß die Fortsetzung dieses und evt. anderer Programme unmöglich ist, und daran sollten Sie sich auch halten.

Um nun den selbst kreierten Alert darzustellen, genügt die Funktionsprozedur DISPLAYALERT:

Taste := DisplayAlert (Typ , Message , Höhe);

Typ	LONGCARD-Wert des Alert-Typs, entweder recoveryAlert oder deadEndAlert.
Message	ADDRESS-Wert für die Adresse des Records oder Strings mit dem Alert-Text.
Höhe	INTEGER-Wert, der die Höhe des Alert-Kastens in Bildpunkten angibt.
Taste	BOOLEAN-Rückgabewert, ist TRUE wenn die linke Maustaste oder FALSE wenn die rechte gedrückt wurde.

Noch eine Bitte: Benutzen Sie Alerts nur, wenn es sich wirklich nicht vermeiden läßt, damit ihr Schrecken und ihre Effektivität gewahrt bleiben! Für weniger wichtige Meldungen sind Requester eine schöne und universelle Alternative.

Beispielprogramm:

```
MODULE AlertDemo;

FROM SYSTEM    IMPORT ADR;
FROM Intuition IMPORT recoveryAlert,DisplayAlert;

TYPE Text=RECORD
       x1    : CARDINAL;
       Zeile1: ARRAY[Ø..45] OF CHAR;
       x2    : CARDINAL;
       Zeile2: ARRAY[Ø..27] OF CHAR;
     END;

VAR Alert: Text;

BEGIN

  WITH Alert DO
    Zeile1:="*Keine Panik, dieser Alert ist nicht echt! ";
    x1:=136; Alert.Zeile1[Ø]:=CHAR(15); Alert.Zeile1[45]:=CHAR(1);
    Zeile2:="*--- Maustaste drücken ---";
    x2:=2Ø6; Alert.Zeile2[Ø]:=CHAR(28); Alert.Zeile2[27]:=CHAR(Ø);
  END;

  IF DisplayAlert(recoveryAlert,ADR(Alert),4Ø) THEN END;

END AlertDemo.
```

3.8 Preferences

Die von Preferences erzeugte Datei »system-configuration« im devs-Ordner befindet sich auf jeder bootfähigen Diskette. In ihr sind alle Informationen enthalten, die Sie mit dem Preferences-Programm festlegen können, z.B. die Workbench-Farben, Ihr Drucker oder auch die Position des Monitor- oder Fernsehbildes. All diese Informationen sind in einem Record zusammengefaßt:

```
Preferences=RECORD
  fontHeight     : UByte                           ; Höhe des Zeichensatzes
  printerPort    : PrinterPort                     ; Druckeranschluß
  baudRate       : CARDINAL                        ; Baudrate für DFÜ
  keyRptSpeed    : TimeVal                         ; Wiederholungsgeschwin-
                                                     digkeit für Tasten
  keyRptDelay    : TimeVal                         ; Zeit, nach der die
                                                     Wiederholung beginnt
  doubleClick    : TimeVal                         ; Zeit für Doppelklick
  pointerMatrix  : ARRAY [0..pointersize-1] OF CARDINAL  ; Spritedaten für Maus-
                                                     zeiger
```

```
xOffset          : Byte                                  ; x-Abstand zum Hotspot
yOffset          : Byte                                  ; y-Abstand zum Hotspot
color17          : CARDINAL                              ; 1.Farbe des Maus-
                                                           zeigers
color18          : CARDINAL                              ; 2.Farbe des Maus-
                                                           zeigers
color19          : CARDINAL                              ; 3.Farbe des Maus-
                                                           zeigers
pointerTicks     : CARDINAL                              ; Pointerticks
colorØ           : CARDINAL                              ; 1.Workbenchfarbe
                                                           (Hintergrund)
color1           : CARDINAL                              ; 2.Workbenchfarbe
                                                           (Vordergrund)
color2           : CARDINAL                              ; 3.Workbenchfarbe
color3           : CARDINAL                              ; 4.Workbenchfarbe
                                                           (Cursor)
viewXOffset      : Byte                                  ; x-Position des
                                                           Monitorbildes
viewYOffset      : Byte                                  ; y-Position des
                                                           Monitorbildes
viewInitX        : INTEGER                               ; Kopie des x-Initial-
                                                           Views
viewInitY        : INTEGER                               ; Kopie des y-Initial-
                                                           Views
enableCLI        : BOOLEAN                               ; CLI erlaubt?
printerTyp       : PrinterType                           ; Druckertyp
printerFilename  : ARRAY [Ø..filenameSize-1] OF CHAR     ; Name des Drucker-
                                                           treibers
printPitch       : CARDINAL                              ; Zeichendichte
printQuality     : CARDINAL                              ; Schriftqualität
printSpacing     : CARDINAL                                Zeilendichte
printLeftMargin  : CARDINAL                              ; linker Rand
printRightMargin : CARDINAL                              ; rechter Rand
printImage       : CARDINAL                              ; Positiv- oder Negativ-
                                                           Bildausdruck
printAspect      : CARDINAL                              ; Art des Grafik-
                                                           ausdrucks
                                                           (horizontal/vertikal)
printShade       : CARDINAL                              ; Farbe des Grafik-
                                                           ausdrucks
printThreshold   : CARDINAL                              ; Auflösung beim Grafik-
                                                           druck
paperSize        : CARDINAL                              ; Seitenformat
paperLength      : CARDINAL                              ; Papierlänge
```

```
paperType        : CARDINAL                              ; Papiertyp
serRWBits        : UByte                                 ; serielle Schreib-/
                                                           Lesebits
serStopBuf       : UByte                                 ; Größe des seriellen
                                                           Puffers
serParShk        : SerParShkSet                          ; Handshaking-Werte
laceWB           : BOOLEAN                               ; Workbench-Interlace?
workName         : ARRAY [Ø..filenameSize-1] OF CHAR     ; Name der Bootdiskette
                                                           (für Intuition)
padding          : ARRAY [Ø..15] OF Byte                 ; für zukünftige
                                                           Versionen
                                                           END;
```

Nun, wenn Sie Lust haben, wollen wir uns gemeinsam an die Geheimnisse dieses Mega-Records wagen:

fontHeight Hier wird der Workbench-Zeichensatz (Topaz 60 oder 80) angegeben, was entweder 60 oder 80 Zeichen pro Zeile heißt. Es sind nur die beiden Konstanten topazSixty (9) und topazEighty (8) zulässig.

printerPort Hier steht, ob Ihr Drucker an der parallelen (parallelPrinter) oder an der seriellen (serialPrinter) Schnittstelle angeschlossen ist.

baudRate Dieses Feld zeigt die Übertragungsgeschwindigkeit in Baud für Datenfernübertragung oder MIDI an. Zulässig sind folgende Konstanten:

1. baud110 (0) für 110 Bauds
2. baud300 (1) für 300 Bauds
3. baud1200 (2) für 1200 Bauds
4. baud2400 (3) für 2400 Bauds
5. baud4800 (4) für 4800 Bauds
6. baud9600 (5) für 9600 Bauds
7. baud19200 (6) für 19200 Bauds
8. baudMidi (7) für MIDI-Anschluß (Anschluß eines MIDI-Instrumentes wie Synthesizer oder Keyboard)

keyRptSpeed In diesem Feld steht die Geschwindigkeit bei der Tastenwiederholung, die ja auch einstellbar ist. Die Werte müssen als TimeVal-Record (aus dem TimerDevice-Definitionsmodul) angegeben werden:

```
TimeVal=RECORD
  secs : LONGCARD ; Sekunden
  micro: LONGCARD ; Mikrosekunden
END;
```

keyRptDelay Hier steht, wie lange eine Taste gedrückt gehalten werden muß, um die Wiederholung einzuschalten. Zum TimeVal-Record siehe keyRptSpeed.

doubleClick Dieses Feld gibt an, wieviel Zeit maximal zwischen den einzelnen Klicken bei einem Doppelklick mit der Maus vergehen darf. Der Wert muß im TimeVal-Format angegeben werden. Dazu siehe keyRptSpeed.

pointerMatrix In diesem Feld steht eine Tabelle mit den Daten des Mauszeigers im Sprite-Format.

xOffset Hier wird der x-Abstand des übrigen Mauszeigers zum Hotspot, dem eigentlichen Klickpunkt, angegeben. Ein Objekt gilt nur als angeklickt, wenn der Hotspot darüber steht.

yOffset Hier steht der y-Abstand zum Hotspot.

pointerTicks Dieses Feld gibt an, wieviele IntuiTicks (1/50stel Sekunden) nötig sind, um eine Koordinate des Mauszeigers zu erhöhen. Im Preferences-Programm können Sie 1, 2 oder 4 angeben, wobei 2 wohl die beste Wahl ist, um genau steuern zu können. Größere Werte sind nicht sinnvoll, da Sie, wenn Sie z.B. den Wert 32768 angeben würden, die Maus mehr als eine Meile (= 1,609 Kilometer!) bewegen müßten, um den Zeiger vom oberen zum unteren Bildrand zu bewegen.

viewXOffset In diesem Feld wird der x-Abstand des Monitor- oder Fernsehbildes zum Punkt (0;0) angegeben.

viewYOffset Hier steht der y-Abstand des Monitor- oder Fernsehbildes zum Punkt (0;0).

viewInitX Hier steht der Standard-x-Abstand des Monitor- oder Fernsehbildes zum Punkt (0;0).

viewInitY Dieses Feld enthält den Standard-y-Abstand des Monitor- oder Fernsehbildes zum Punkt (0;0).

enableCLI Dieser BOOLEAN-Wert gibt an, ob das CLI-Icon sichtbar ist (TRUE) oder nicht (FALSE). Es darf am Sinn dieses Feldes

gezweifelt werden. Ich für meine Verhältnisse arbeite mehr mit dem CLI als mit der Workbench, aber das nur nebenbei.

printerType In diesem Feld steht der wievielte Druckertyp, der von den angebotenen gewählt wurde. Zulässig sind, wenn alle Workbench-Druckertreiber im »DEVS/PRINTERS«-Verzeichnis sind, folgende Werte (gilt wahrscheinlich nur für Version 1.2, da 1.3 bzw. 1.4 neue Druckertreiber haben sollen):

1.	customName	: eigener Druckertreiber
2.	alphaP101	: Alpha P-101
3.	brother15XL	: Brother HR-15XL
4.	cbmMps1000	: Commodore MPS-1000
5.	diab630	: Diabolo 630
6.	diabAdvD2 5	: Diabolo Advanced D-25
7.	diabC150	: Diabolo C-150
8.	epson	: Epson FX-80
9.	epsonJX80	: Epson JX-80
10.	kimate20	: Okimate 20
11.	QumeLP20	: Qume LP-20
12.	hpLaserJet	: HP Laserjet
13.	hpLaserjetPlus	: HP Laserjet Plus

Wenn Sie allerdings nur zwei Drucker, z.B. »custom« und »Epson_JX-80« zur Auswahl haben und den Epson-Drucker gewählt haben, so steht in diesem Feld die Ordungszahl 1, also die Nummer, die eigentlich dem Alpha P-101 zusteht. Schade, das macht die an sich einfache Druckertyp-Abfrage etwas komplizierter.

Selbstverständlich können Sie auch kompatible Drucker angeben. Für den Seikosha MP-1300AI z.B. (falls den außer mir noch jemand besitzen sollte) sollten Sie den Epson JX-80-Treiber wählen.

printerFilename Hier steht der Dateiname Ihres Druckertreibers.

printPitch Hier wird die Zeichendichte beim Ausdruck angegeben. Zulässig sind folgende Konstanten:

pica (0H): normale Pica-Schrift, normalerweise 80 Zeichen/Zeile
elite (0400H): Elite-Schrift, bei den meisten Druckern 96 Zeichen/Zeile
fine (0800H): Schmalschrift, normalerweise 136 Zeichen/Zeile

printQuality Hier steht die Standard-Schriftqualität beim (Text-)Ausdruck. Zulässig sind:

draft (0H): normale EDV-Schrift
letter (0100H): Brief-(LQ-)Schrift oder Beinahe-Brief-(NLQ-) Schrift

printSpacing — Dieses Feld gibt den Zeilenabstand an. Auftreten können folgende Werte:

sixLPI (0H): 6 Zeilen pro Zoll
eightLPI (0200H): 8 Zeilen pro Zoll

printLeftMargin — In diesem Feld steht die Größe des linken Randes in Zeichen.

printRightMargin — Hier wird die Größe des rechten Randes in Zeichen angegeben.

printImage — Dieses Feld ist dazu da, die Art des (Grafik-) Ausdrucks anzugeben:

imagePositive (0): Das Bild wird so gedruckt, wie es auf dem Bildschirm steht.
imageNegative (1): Das Bild wird negativ gedruckt.
Dieses Feld ist nur für Schwarz/Weiß-Ausdrucke von Bedeutung.

printAspect — Hier steht, ob das Bild normal (aspectHoriz, 0) oder um 90 Grad gedreht (aspectVert, 1) gedruckt werden soll.

printShade — In diesem Feld wird die Art des Grafikausdrucks angegeben:

shadeBW (#0): Schwarz/Weiß-Ausdruck (nicht sehr repräsentativ)

shadeGreyScale (#1): Graustufen-Ausdruck, jede Farbe bekommt beim Ausdruck einen hellen oder dunklen Grauton (für Besitzer von Schwarz/Weiß-Druckern ideal)

shadeColor (#2): Farbdruck, natürlich nur für Farbdrucker. Ein empfehlenswerter Treiber für diese Option ist übrigens der Epson JX-80, sofern Ihr Drucker kompatibel ist.

printThreshold — Hier steht der Wert, der angibt, welche Farbstufen beim Schwarz-/Weiß-Druck schwarz gedruckt werden. Je größer dieser Wert ist, um so dunkler werden die Farben, die als schwarz gedruckt werden sollen.

paperSize — In diesem Feld steht der Wert, der die Größe Ihres Papiers angibt. Folgende Werte sind nur zulässig und dürfen bei eventueller Modifikation von Ihnen benutzt werden:

usLetter (0H) — US-Briefformat; 8,5 Zoll breit und 11 Zoll hoch.

usLegal (010H) US-Standardformat; 8,5 Zoll breit und 14 Zoll hoch.

nTractor (020H) schmaler Zugtraktor; 9,5 Zoll breit und 11 Zoll hoch (bei normalen 80 Zeichen/Zeile-Druckern (Pica) ist diese Einstellung ganz gut).

wTractor (030H) breiter Zugtraktor; 14,875 Zoll breit und 11 Zoll hoch.

custom (040H) eigenes Format.

paperLength Dieses Feld zeigt die Papierlänge in Zeilen an.

paperType In diesem Feld steht der Papiertyp, entweder Endlos- (fanfold, 0H) oder Einzelblatt-Papier (single, 80H).

serRWBits Hier stehen die für serielle Übertragung nötigen Bits, entweder Lesebits (readBits, 0F0H) oder Schreibbits (writeBits, 0FH).

serStopBuf Dieses Record-Feld gibt die Stop-Bits und den Puffer für serielle Übertragung an. Folgende Werte dürfen eingetragen werden:

stopBits (0F0H)	: serielle Stop-Bits
bufSizeBits (0FH)	: Bits für die Größe des Puffers
buf512 (0)	: 512 Byte Puffer
buf1024 (1)	: 1024 Byte Puffer
buf2048 (2)	: 2048 Byte Puffer
buf4096 (3)	: 4096 Byte Puffer
buf8000 (4)	: 8000 Byte Puffer
buf16000 (5)	: 16 000 Byte Puffer

serParShk Hier wird der Handshaking-Wert (Kontrolle des Datenflusses) für serielle Übertragung angegeben. Zulässig sind die Werte der Menge SerParShkSet:

shakeXon: xOn/xOff-Wert
shakeRts: RTS/CTS-Wert
shakeNone: kein Handshaking
sps3: fürs System reserviert
parityNone: keine Parität (Parität = Erkennung von Übertragungsfehlern)
parityEven: gerade Parität
parityOdd: ungerade Parität

laceWB Dieses BOOLEAN-Feld dient dazu, den Interlace-Modus für die Workbench anzuzeigen. TRUE bedeutet Workbench-Interlace an, FALSE hingegen heißt, daß die Workbench normal dargestellt wird. Zwar wird bei der Benutzung dieses Feldes die Wirkung erst

nach einem Reset sichtbar, doch inzwischen sind wohl schon genug Programme in Zeitschriften veröffentlicht worden oder als Public-Domain in Umlauf, die einen sofortigen Workbench-Interlace-Modus bewirken.

workName	Hier steht der Name der Diskette, von der gebootet wurde. Dieser Diskette wird das Device SYS: zugeordnet.

3.8.1 Preferences-Praxis

Im vorherigen Kapitel haben wir den Preferences-Record ausführlich (und doch wohl ausreichend?) beschrieben, doch was nützen uns diese Informationen, wenn wir nichts damit anfangen können? Sie haben es erraten: nichts. Um die Preferences zu holen, gibt es zwei verschiedene Arten: 1. Wir holen uns die Preferences der Diskette, von der gebootet wurde oder 2. Wir holen uns die aktuellen Preferences aus dem Speicher (z.B. wenn Sie beim Preferences-Programm vor Verlassen Ihre Werte mit USE und nicht mit SAVE festgehalten haben, können diese mehr oder minder verschieden von den Startwerten sein). Zu 1.: Hierzu existiert der Befehl GETDEFPREFS:

GetDefPrefs (MeinePreferences , Bytes);

MeinePreferences	PreferencesPtr auf den Record, in den die Preferences eingelesen werden sollen.
Bytes	INTEGER-Wert, der die Anzahl der Bytes, die eingelesen werden sollen, angibt.

Zu 2.: Hierfür gibt es den Befehl GETPREFS:

GetPrefs (MeinPreferences , Bytes);

MeinePreferences	PreferencesPtr auf den Record, in den die Preferences eingelesen werden sollen.
Bytes	INTEGER-Wert der Anzahl der Bytes, die eingelesen werden sollen.

Wenn Sie dann die Preferences in Ihren eigenen Record kopiert und nach Herzenslust verändert haben, kann es manchmal ganz sinnvoll sein, sie zurückzuschreiben. Diese Aufgabe übernimmt das SETPREFS-Kommando:

SetPrefs (MeinePreferences , Bytes , Disk);

MeinePreferences	PreferencesPtr auf Ihre (möglicherweise veränderten) Preferences.
Bytes	Anzahl der Bytes, die geschrieben werden sollen (INTEGER).
Disk	BOOLEAN-Wert, der angibt, ob die neuen Preferences auf Diskette (TRUE) oder nur in den Speicher (FALSE) geschrieben werden sollen.

Soviel zu den Preferences. Was nun folgt, ist ein Beispielprogramm, welches den Umgang mit dem Preferences-Record verdeutlichen soll:

```
MODULE PreferencesDemo;

FROM SYSTEM    IMPORT ADR;
FROM Intuition IMPORT GetPrefs,SetPrefs,Preferences;
FROM Dos       IMPORT Delay;

VAR MyPrefs     : Preferences;
    oldx,oldy   : [-128..127];
    i           : INTEGER;

BEGIN
  GetPrefs(ADR(MyPrefs),SIZE(MyPrefs)); (* Preferences holen *)

  oldx:=MyPrefs.viewXOffset;            (* x-Offset sichern *)
  oldy:=MyPrefs.viewYOffset;            (* y-Offset sichern *)

  MyPrefs.viewYOffset:=Ø;               (* y-Offset auf Ø *)
  FOR i:=Ø TO 5Ø DO
    MyPrefs.viewXOffset:=i;             (* kleine Animation *)
    SetPrefs(ADR(MyPrefs),SIZE(MyPrefs),FALSE);
  END;

  FOR i:=Ø TO 5Ø DO
    MyPrefs.viewYOffset:=i;
    SetPrefs(ADR(MyPrefs),SIZE(MyPrefs),FALSE);
  END;

  FOR i:=Ø TO 5Ø DO
    MyPrefs.viewXOffset:=i;
    SetPrefs(ADR(MyPrefs),SIZE(MyPrefs),FALSE);
  END;

  FOR i:=Ø TO 5Ø DO
    MyPrefs.viewYOffset:=i;
    SetPrefs(ADR(MyPrefs),SIZE(MyPrefs),FALSE);
  END;
```

```
  Delay(1ØØ);

  MyPrefs.viewXOffset:=oldx;
  MyPrefs.viewYOffset:=oldy;
  SetPrefs(ADR(MyPrefs),SIZE(MyPrefs),FALSE);

END PreferencesDemo.
```

3.9 Images

Kommen wir nun erstmal zu den in diesem Buch schon vielzitierten Images. Doch was sind Images überhaupt? Bilder, oder besser Bildchen lautet die Antwort. Wenn Sie bei einem Gadget oder einem Menüpunkt anstelle eines Textes ein kleines Bild darstellen möchten, so sollten Sie es mal mit Images versuchen.

3.9.1 Berechnung der Image-Werte

Um ein Image, das etwas komplizierter als ein ausgefüllter (oder völlig leerer) Block ist, würde ich Ihnen raten, dieses erst auf kariertem Papier vorzuzeichnen und die entsprechenden Zahlenwerte auszurechnen. Dabei verfahren Sie am besten wie folgt:

1. Zeichnen Sie ein Raster mit den Ausmaßen des Images; sollten Sie noch keine klare Vorstellung von Ihrem Image haben (bitte nicht mißverstehen), so empfiehlt es sich, erstmal eine oder mehrere Vorzeichnungen zu machen.
2. Zeichnen Sie Spalten mit je vier Kästchen ein und schreiben Sie oberhalb jeder Spalte genau über den vier Kästchen die Zahlen 8, 4, 2 und 1 (Zweierpotenzen).
3. Nun können Sie das Image definieren, indem Sie es in Ihrem Raster darstellen. Für jeden gesetzten Punkt sollten Sie ein Kästchen ausfüllen (oder einfach ein Kreuzchen hineinzeichnen, wie beim Spiel »Käsekästchen«).
4. Jetzt darf gerechnet werden: zählen Sie bitte die Werte der einzelnen Spalten zusammen indem Sie überall dort, wo in einer Spalte und einer Reihe (!), die Kennzahlen (die Zweierpotenzen am oberen Rand) zueinander addieren. Die dabei herausgekommenen Werte schreiben Sie (am besten im Hexadezimalsystem) dann in eine zweite Tabelle, ebenfalls in Spalten aufgeteilt. Je vier in einer Reihe stehenden Werte fassen Sie nun zu einer einzigen Hex-Zahl zusammen (damit ein Zwei-Byte-Wert oder Word herauskommt). Darauf können Sie dann die einzelnen Reihen (in denen jetzt pro Reihe eine Zahl steht) in Ihren Datenblock eintragen.

3.9.2 Dateneingabe – ein neues Prinzip

Es ist einfach witzlos, die Image-Daten irgendwie in einen Array zu pressen wie die Border-Werte, besonders bei komplexen Bildern schreibt man sich die Finger wund. Daher möchte ich ein neues Prinzip der Dateneingabe einführen: Speichernahe Tabellen mit INLINE (bzw. CODE bei TDI). Bei dieser Methode werden die erhaltenen Hexadezimalzahlen (die ja alle die Größe eines Words, das auch nur von INLINE angenommen wird, haben) einfach per INLINE in den Speicher geschrieben. Am besten ist es wohl, eine Prozedur mit sämtlichen INLINE-Anweisungen zu schreiben. Vor so einer Prozedur muß beim M2-Compiler die Anweisung »$E–«, beim TDI-System »$P–«, dem am Prozedurende ein »$P+« folgen muß. (Durch diesen Schalter erreicht man das Ausschalten eines speziellen Ein-/Ausstiegscode, den der Compiler vor jede bzw. nach jeder Prozedur anhängt. In unserem Fall wollen wir allerdings die reinen Daten.

Nach dem gleichen Prinzip können Sie mit Screen- oder Fensterstrukturen verfahren – Titel und Flags können Sie ja nachträglich hineinbringen (oder Sie kennen die Hex-Werte der einzelnen Flags – dann können Sie dieses Feld bereits per INLINE ausfüllen. Sind Sie daran interessiert, sollten Sie im Anhang B nachschauen). Diese Methode ist vor allen Dingen zu empfehlen, wenn Sie Ihren Quell- und Objektcode möglichst kompakt halten möchten, sonst halte ich dieses Prinzip für Anwendungen außerhalb von Datenblöcken Border- oder Image-Datenblöcken für unangebracht. Borders sollten Sie allerdings unbedingt in diesem Format angeben (siehe Bild Seite 125).

3.9.3 Alles Weitere über Images

Der Rest der Image-Daten, also Dinge wie Position, Farbe usw. ist nicht im Datenblock angegeben. Folglich muß wieder ein Record herhalten, Image:

```
Image=RECORD
  leftEdge  : INTEGER        ; x-Koordinate der linken, oberen Ecke des Images
  topEdge   : INTEGER        ; y-Koordinate der linken, oberen Ecke des Images
  width     : INTEGER        ; Breite des Images
  height    : INTEGER        ; Höhe des Images
  depth     : INTEGER        ; Anzahl der Bitplanes des Images
                               (siehe NewScreen-Record)
  imageData : ADDRESS        ; Adresse des Datenblocks (die INLINE-Prozedur)
  planePick : SET OF [Ø..7]; Angabe der Screen-Bitplanes
  planeOnOff: SET OF [Ø..7]; Zustand der übrigen Planes
  nextImage : ImagePtr       ; Zeiger auf das nächste Image
END;
```

leftEdge+topEdge Diese Werte sind relativ zum Objekt, in dem das Image dargestellt werden soll!

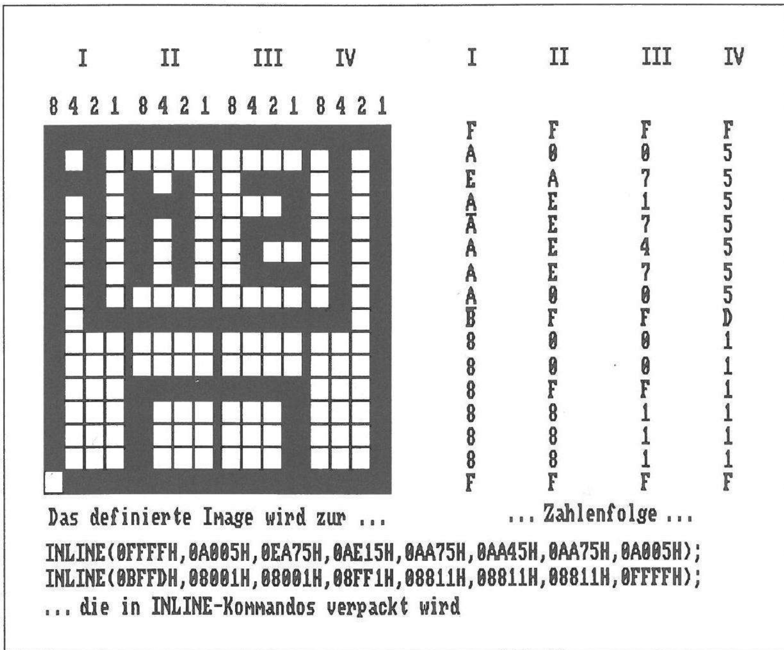


Bild 3.3: Beispiel für die Definition eines Images

planePick Hier wird angegeben, in welche Bitplanes des Screens das Image gezeichnet wird. Zum besseren Verständnis empfehle ich Ihnen, etwas vorzugreifen und das Kapitel über Bitplanes im GRAPHICS-Teil durchzulesen. Da dieses Feld vom Typ SET ist, lassen sich auch mehrere Bitplanes angeben, um ein mehrfarbiges Image zu generieren. Dafür müssen Sie aber die Werte, die in der zweiten, dritten oder was für einer Plane auch immer gezeichnet werden sollen, an den Datenblock angehängt werden.

planeOnOff Dieses Feld ist dazu da, anzugeben, was mit den übrigen Bitplanes, die nicht in planePick angegeben sind, passieren soll.

Bevor wir zu den Zeichenbefehlen von Intuition kommen, sei noch gesagt, daß für die Felder planePick und planeOnOff folgende Werte sinnvoll und damit erlaubt sind:

0: keine Planes sollen gesetzt werden

1: Plane 0 soll gesetzt werden

2: Plane 1 soll gesetzt werden

3: Planes 0 und 1 sollen gesetzt werden

4: Plane 2 soll gesetzt werden

5: Planes 0 und 2 sollen gesetzt werden

6: Planes 1 und 2 sollen gesetzt werden

7: Planes 0, 1 und 2 sollen gesetzt werden

Nun wäre doch soweit alles geklärt – wenn da nicht so eine dumme Kleinigkeit den Umgang mit Images etwas komplizieren würde: Die Image-Daten müssen im CHIP-Ram, den untersten 512 Kbyte liegen! Dummerweise legt mein Modula-2-Compiler (M2Amiga) alle Daten ins Fast-Ram (wohl denen, die einen Lattice-C-Compiler besitzen und damit programmieren ...). Doch wir müssen eigentlich nur ein kleines Speicherkopierprogramm schreiben, welches unsere Image-Daten ins Chip-Ram, das wir vorher mit ALLOCMEM (erinnern Sie sich ans DOS-Kapitel? Doch auf diesen Befehl soll im Kapitel über EXEC detailliert eingegangen werden) bereitgestellt haben, dupliziert. Im Demoprogramm zu 3.10.4 finden Sie so eines (die Prozedur ChipCopy), so daß es an dieser Stelle nicht abgedruckt zu werden braucht.

3.9.4 Die Intuition-Zeichenbefehle

Ja, Intuition stellt auch Zeichenbefehle zur Verfügung. Diese verblassen zwar vor der Vielfalt der GRAPHICS-Zeichenbefehle, doch für eine gute Benutzeroberfläche reichen sie vollkommen aus. Fangen wir an: Um die eben besprochenen Images auch zu zeichnen, gibt es den Befehl DRAWIMAGE:

DrawImage (MeinRastPort , MeinImage , OffsetX , OffsetY);

MeinRastPort	RastPortPtr auf den RastPort des Fensters (wie man an diesen kommt, erfahren Sie im GRAPHICS-Kapitel.)
MeinImage	ImagePtr auf den Image-Record des zu zeichnenden Bildes.
OffsetX	INTEGER-Wert, der die Zahl, der zum Feld leftEdge des Image-Records dazuaddiert werden soll, angibt. Möchte man das Image allein in einem Fenster (nicht in einem Gadget o.ä.) stehen haben, ist es sinnvoll, diesen und den nächsten Parameter für die absolute Position zu nutzen.
OffsetY	Genau das gleiche wie »OffsetX«, nur für den y-Wert (wird zum Feld topEdge addiert).

Auch zum reinen Zeichnen eines Borders existiert ein Befehl, denn manchmal kommt der Wunsch auf, einen Border um ein Image oder einfach so zu zeichnen, ohne Gadget o.ä. Besagter Befehl heißt DRAWBORDER:

DrawBorder (MeinRastPort , MeinBorder , OffsetX , OffsetY);

MeinRastPort	RastPortPtr des Windows, in dem der Border erscheinen soll. Da das Thema RastPort in das GRAPHICS-Kapitel gehört, bitte ich Sie, dort nachzuschlagen.
MeinBorder	BorderPtr auf den Border-Record des Borders, der gezeichnet werden soll.
OffsetX	INTEGER-Wert, der eine eventuelle Zahl, die zum Inhalt des Feldes leftEdge addiert werden soll. Meist ist es sinnvoll, hier die absolute Position anzugeben und die Felder leftEdge und topEdge auf 0 zu setzen.
OffsetY	Gegenstück zu »OffsetX«, nur für die y-Koordinate bzw. das Feld topEdge gültig.

Und als letztes wird der PRINTITEXT-Befehl besprochen. Wie man sich vielleicht schon denken kann, dient er dazu, IntuiTexte auszugeben:

PrintIText (MeinRastPort , MeinText , OffsetX , OffsetY);

MeinRastPort	Siehe DRAWIMAGE.
MeinText	IntuiTextPtr auf den IntuiText-Record des ersten Textes, der ausgegeben werden soll. Möchten Sie mehrere Texte hintereinander schreiben, so können Sie die verschiedenen Record über das Feld nextText verbinden.
OffsetX	INTEGER-Zahl, die einen Wert, der zu dem Inhalt des Feldes leftEdge der IntuiText-Struktur addiert wird, angibt. Meist wird hier die absolute Position des Textes angegeben und die Felder leftEdge und topEdge auf 0 gesetzt.
OffsetY	Entspricht dem Parameter »OffsetX« mit Ausnahme der Tatsache, daß sich dieser Wert auf die y-Koordinate bzw. das Feld topEdge bezieht.

Zum Abschluß soll, wie immer, ein Beispielprogramm folgen, das allerdings nur den Umgang mit Images erläutert, Borders und IntuiTexte dürften Ihnen aber keine Schwierigkeiten mehr machen:

```
MODULE ImageDemo;

FROM SYSTEM    IMPORT ADR,ADDRESS,INLINE;
FROM Intuition IMPORT DrawImage,Image,NewWindow,WindowPtr,OpenWindow,
                      CloseWindow,IDCMPFlags,IDCMPFlagSet,WindowFlags,
                      WindowFlagSet,ScreenFlags,ScreenFlagSet;
FROM Exec      IMPORT AllocMem,MemReqs,MemReqSet,FreeMem;
FROM Dos       IMPORT Delay;

TYPE CardPtr = POINTER TO CARDINAL;

VAR MeinImage  : Image;
    MeinWindow : WindowPtr;
    WindowDaten: NewWindow;
    ChipImage  : CardPtr;
    i,j        : INTEGER;

(* Hier steht die besagte Speicherblock-Kopierroutine *)
PROCEDURE ChipCopy(source: CardPtr; VAR dest: CardPtr; size: LONGCARD);
VAR ChipPtr: CardPtr;
    copied : LONGCARD;
  BEGIN
    ChipPtr:=AllocMem(size,MemReqSet{chip});
    dest:=ChipPtr;
    copied:=Ø;
    REPEAT
      ChipPtr^ :=source^ ;
      INC(ChipPtr,2); INC(source,2); INC(copied,2);
    UNTIL copied=size;
  END ChipCopy;

PROCEDURE ImageDaten; (* $E- *)
  BEGIN
    INLINE(ØFFFFH,ØAØØ5H,ØEA75H,ØAE15H,ØAA75H,ØAA45H,ØAA75H,ØAØØ5H);
    INLINE(ØBFFDH,Ø8ØØ1H,Ø8ØØ1H,Ø8FF1H,Ø8811H,Ø8811H,Ø8811H,Ø7FFFH);
  END ImageDaten;

BEGIN
  ChipImage:=NIL;
  ChipCopy(ADR(ImageDaten),ChipImage,32);

  WITH MeinImage DO
    leftEdge:=Ø; topEdge:=Ø; width:=16; height:=16; depth:=1;
    imageData:=ChipImage; planePick:=1; planeOnOff:=Ø; nextImage:=NIL;
  END;
```

```
  WITH WindowDaten DO
    leftEdge:=0; topEdge:=0; width:=640; height:=256; detailPen:=0;
    blockPen:=1; idcmpFlags:=IDCMPFlagSet{}; flags:=WindowFlagSet{borderless};
    firstGadget:=NIL; checkMark:=NIL; title:=NIL; screen:=NIL; bitMap:=NIL;
    type:=ScreenFlagSet{wbenchScreen};
  END;

  MeinWindow:=OpenWindow(WindowDaten);

  FOR i:=0 TO 640 BY 16 DO
    FOR j:=0 TO 256 BY 16 DO
      DrawImage(MeinWindow^.rPort,ADR(MeinImage),i,j);
    END;
  END;

  Delay(200);

  CloseWindow(MeinWindow);
  FreeMem(ChipImage,32);
END ImageDemo.
```

3.10 Intuition-Richtlinien

Es gibt gewisse Richtlinien, die Commodore den Intuition-Programmierern setzt, damit alle Programme einen hohen Standard an Benutzerfreundlichkeit erreichen. Zwar hält sich nicht jeder daran, aber dennoch möchte ich diese Richtlinien aufführen:

Zu den Menüs:

1. Wenn ein Menüpunkt im Programm momentan keine Bedeutung hat, sollte er mit OFFMENU abgeschaltet werden, so daß der Benutzer keine Möglichkeit hat, diesen anzuwählen. Sie sollten nicht stets dem Benutzer etwas anbieten, das er zwar anwählen kann, aber das keine Wirkung hat.

2. Sollte Ihr Programm mehrere Windows haben, jedes mit unterschiedlichen Menüs, so wäre es nett von Ihnen, diese farblich zu unterscheiden.

3. Benötigt Ihr Programm ein »Project«- bzw. Dateimenü (solche Menüs sollten übrigens immer ganz links in der Menüleiste stehen), so sollten folgende Menüpunkte darin vorkommen (wenn möglich, in dieser Reihenfolge):

NEW (Neue Datei)	Eine neue Datei anlegen
OPEN (Alte Datei öffnen)	Eine bereits bestehende Datei laden
SAVE (Speichern)	Die Datei unter dem Namen speichern, unter dem sie geladen wurde

SAVE AS (Speichern als)	Die Datei unter einem neuen Namen speichern
PRINT (Drucken)	Die Datei ausdrucken
PRINT AS (Drucken als)	Teile der Datei ausdrucken oder neue Druckparameter wählen
QUIT (Ende)	Beenden des Programms, wenn eine Datei verändert wurde, mit Sicherheitsabfrage, ob der Benutzer die neue Datei speichern möchte

4. Wenn Ihr Programm ein »Edit«-Menü (zur Textmodifizierung) braucht (z.B. ein Editor), so sollte dieses Menü folgende Punkte beinhalten:

UNDO (Rückgängig machen)	Macht die letzte Operation rückgängig.
CUT (Ausschneiden)	Den angewählten Bereich in einen zusätzlichen Speicher (z.B. das ClipBoard-Device) schieben (das Original wird gelöscht.)
COPY (Kopieren)	Den angewählten Bereich in einen Spezialspeicher wie das ClipBoard-Device kopieren (das Original bleibt stehen.)
PASTE (Einfügen)	Den Inhalt des »CUT«- bzw. »COPY«-Speichers an einer bestimmten Stelle einfügen.
ERASE (Löschen)	Den angewählten Bereich löschen, ohne ihn zwischenzuspeichern.

Zu den Gadgets:

1. Verwendet Ihr Programm Gadgets in Requestern oder Windows, so sollte das Gadget, welches für den »sicheren« Abschluß der Handlung steht (meistens Gadgets wie »NEIN« oder »ABBRUCH«), besonders hervorgehoben werden.
2. Gadgets, die momentan keine Funktion haben, sollten mit OFFGADGET ausgeschaltet werden.
3. Gadgets sollten optimal und übersichtlich plaziert werden und dürfen sich niemals überlappen, damit der Benutzer sofort den »Durchblick« hat, was er eigentlich anwählen kann oder soll.

Zu den Requestern:

1. Es sollte stets die Möglichkeit bestehen, den Requester zu verlassen, ohne eine Aktion durchzuführen.
2. Bei Requestern, die nur zwei Auswahlmöglichkeiten anbieten, sollte das Gadget, das für ein sicheres Verlassen des Requesters sorgt, auf der linken Seite plaziert sein.

Zu den Shortcuts:

1. Der Benutzer sollte in der Lage sein, ein Shortcut zu betätigen, in dem der Kleine Finger der rechten Hand (bzw. der Daumen der linken bei Linkshändern) die rechte Amiga-Taste drückt und ein anderer Finger die Shortcut-Taste betätigt. Dieser Vorgang sollte bequem erfolgen. Die <ESC>-Taste als Shortcut zu wählen ist verhältnismäßig ungünstig, da nicht jeder so große Hände hat, daß er das Shortcut mit einer Hand bequem anwählen kann.

Zu der Maus:

1. Die linke Maustaste sollte für das Anklicken und die rechte für die Menüauswahl reserviert bleiben.

Verschiedenes:

1. Ihr Programm sollte am Anfang einen OPENWORKBENCH-Befehl stehen haben, selbst wenn die Workbench nicht geschlossen war. Die Workbench sollte, wenn der Speicherplatz es erlaubt, stets geöffnet sein. Sie muß aber unbedingt geöffnet werden, wenn irgendein Screen geschlossen wird (es könnte ja der letzte Screen sein. Wenn dann keine Workbench geöffnet ist, kann der Benutzer nur neu booten).

2. Definieren Sie Ihren eigenen Mauszeiger, sollten Sie folgende Farbdefinitionen beachten:
   - die erste Farbe (Hintergrund) sollte transparent sein.
   - die zweite Farbe (Hardware-Register 17) sollte von mittlerer Intensität sein.
   - die dritte Farbe (Hardware-Register 18) sollte von geringer Intensität sein (möglichst dunkel).
   - die vierte Farbe (Hardware-Register 19) sollte von hoher Intensität sein (möglichst hell).

   Der Zeiger sollte entweder von der zweiten oder vierten Farbe umrandet sein.

Noch etwas:

1. Sie sollten ordentliche und schön anzusehende Gadgets, Menüs und Requester darstellen, die einfach und elegant wirken, denn das Arbeiten mit Ihrem Programm sollte jedesmal Spaß machen, wobei ein Genuß für das Auge auch einen Teil beiträgt.

2. Versuchen Sie, das Aussehen Ihrer Programme möglichst einheitlich und prächtig zu gestalten und sie voll in Intuition einzubinden. Aber geben Sie Ihren Programmen niemals ein »kryptisches« oder unergründliches Aussehen. Machen Sie es einfach »unvergeßlich wunderbar«.

Für die manchmal etwas unglücklichen Formulierungen muß ich mich entschuldigen, aber ich glaube, sie geben die im »Intuition-Reference-Manual« ausgedrückten Sachverhalte unverfälscht wieder.

Kapitel 4

GRAPHICS – allgemein

GRAPHICS - der Name verheißt auf jeden Fall einiges. Doch was steckt dahinter? Ganz einfach, wie Sie möglicherweise schon vermutet haben, übernimmt GRAPHICS die gesamte Grafiksteuerung und stellt Befehle zum Zeichnen bestimmter Objekte, zum Setzen von Farben und für vieles mehr zur Verfügung. Auch Intuitiony z.B. benutzt die Grafik-Systemroutinen, um Fensterrahmen, Gadgets, Icons und andere grafische Elemente zu zeichnen (Intuition ist schließlich eine grafische Benutzeroberfläche). Wie das DOS oder Intuition sind auch die GRAPHICS-Befehle in einer Library geordnet, die beim M2Amiga-Compiler, wie immer, nicht geöffnet werden braucht. Als TDI- oder Benchmark-Benutzer möchte ich Sie darauf aufmerksam machen, daß der Library-Zeiger »GfxBase« heißen sollte. Das ist in gewisser Weise genormt und fast alle C- und Assembler-Programmierer halten sich daran.

4.1 Die Funktionsweise von GRAPHICS

Aber wie arbeitet GRAPHICS nun? Die Antwort auf diese Frage ist recht komplex, aber ein Stück Theorie, das Sie wissen sollten, denn dann fällt Ihnen das Verständis vieler Sachverhalte, die mit GRAPHICS zu tun haben, erheblich leichter.

GRAPHICS nutzt einen großen RAM-Bereich, eine sogenannte Bitmap, als Zeichenebene. Auf dieser Bitmap wird genau verzeichnet, welcher Punkt gesetzt (gesetztes Bit) oder gelöscht (nicht gesetztes Bit) ist. Öffnen Sie einen Screen und/oder ein Window, wird für diesen (bzw. dieses) automatisch eine Bitmap zur Verfügung gestellt.

Die Bitmap besteht aus mehreren Bitplanes, die die einzelnen Schichten der Bitmap bilden, je nachdem, wieviele Farben Sie wünschen. In x Bitplanes können Sie 2 hoch x Farben unterbringen, mit der Ausnahme des HAM-Modus, doch auf den kommen wir noch zu sprechen. Um die Farbe eines Punktes herauszubekommen, müssen alle Bit-Werte dieses Punktes zusammengenommen werden. Der dabei herauskommende Wert gibt die Nummer des Farbregisters an.

Es gibt unter anderem zwei wichtige Schnittstellen zum Programmierer, um ihm den Zugriff auf GRAPHICS zu ermöglichen. Welche er benutzt, kommt drauf an, auf welche GRAPHICS-Einheit er zugreifen möchte. Da wäre zum Einen der RastPort, der schon bei den Intuition-Zeichenbefehlen erwähnt wurde. Er wird auch bei den GRAPHICS-Zeichenbefehlen oft gebraucht. Über ihn kommen Ihre Punkte in die Bitmap. Dann ist noch der ViewPort zu erwähnen. Er steuert, wie der Bildschirm auszusehen hat. Dazu gehören Dinge wie Auflösung, Darstellungsmodi, Copperlisten (denen das gesamte Kapitel 4 gewidmet ist) und nicht zuletzt Farben. Im Prinzip steht er über dem RastPort, doch sind beide gleich wichtig. Merke aber: jedes Fenster hat einen RastPort, doch nur jeder Screen einen ViewPort, weil jeder Screen eine eigene Auflösung und eigene Farben haben kann, ein Window ist aber an die Auflösung und die Farben seines Screens gebunden.

Neben den großen beiden Ports gibt es noch ein weiteres, wichtiges Element: den RasInfo. Dieser ist dafür da, Informationen zwischen RastPort und Bitmap auszutauschen.

4.2 Wie kommt man an RastPort und ViewPort?

Alles im letzten Abschnitt Besprochene würde nicht von Nutzen sein, wenn man nicht an die beiden Ports heranzukommen wüßte. Aber wer noch den Screen- und Window-Record aus dem Intuition-Kapitel im Gedächtnis hat, könnte vielleicht schon wissen, wie. Na, ganz einfach: man holt sich den RastPort einfach aus dem Window- oder Screen-Record und den ViewPort aus der Screen-Struktur, da Intuition diese Felder automatisch initialisiert. So einfach ist das.

Im Klartext: Um den RastPort eines Windows zu bekommen, schreiben Sie

MeinRastPort:=MeinWindow^.rPort

wobei »MeinRastPort« vom Typ RastPortPtr und »MeinWindow« vom Typ WindowPtr sein muß.

Möchten Sie einen Zeiger auf den RastPort eines Screens haben, sollten Sie folgendes Kommando benutzen:

MeinRastPort:=ADR(MeinScreen^.rastPort);

Für den ViewPort gilt das gleiche, nur muß man auf die Screen-Struktur zugreifen:

MeinViewPort:=MeinScreen^.viewPort

oder bei Windows

MeinViewPort:=MeinWindow^.wScreen^.viewPort

wobei »MeinScreen« vom Typ ScreenPtr (bzw. »MeinWindow« vom Typ WindowPtr) und »MeinViewPort« vom Typ ViewPort sein muß. Achtung: »MeinViewPort« darf kein Zeiger sein, denn der ViewPort steht wirklich in der Screen-Struktur, nicht nur ein Zeiger auf diesen!

Eine ausführlich Beschreibung der verschiedenen Records soll folgen.

4.2.1 Der RastPort-Record

```
RastPort=RECORD
  layer       : LayerPtr                ; Zeiger auf das Layer des RastPorts
                                          (kommt noch...)
  bitMap      : BitMapPtr               ; Zeiger auf die Bitmap
  areaPtrn    : ADDRESS                 ; Füllmuster für AREAs
  tmpRas      : TmpRasPtr               ; Zeiger auf temporäres Raster
  areaInfo    : AreaInfoPtr             ; Zeiger auf AreaInfo
  gelsInfo    : GelsInfoPtr             ; Zeiger auf GelsInfo
  mask        : UByte                   ; Maske
  fgPen       : UByte                   ; Vordergrundfarbe
  bgPen       : UByte                   ; Hintergrundfarbe
  aOlPen      : UByte                   ; AREA-Outline-Pen
  drawMode    : DrawModeSet             ; aktueller Zeichenmodus
  areaPtSz    : UByte                   ; 2 hoch areaPtSz Doppelbytes
                                          Speicher für AREA-Füllmuster
  linPatCnt   : UByte                   ; für das System
  dummy       : BYTE                    ; Dummy-Variable
  flags       : RastPortFlagSet         ; RastPort-Flags
  linePtrn    : CARDINAL                ; aktuelles Linienmuster
  x           : INTEGER                 ; aktuelle x-Koordinate
  y           : INTEGER                 ; aktuelle y-Koordinate
  minterms    : ARRAY[Ø..7] OF BYTE     ; aktuelle Minterms (interner Gebrauch)
  penWidth    : INTEGER                 ; Breite des Cursors
                                          (interner Gebrauch)
  penHeight   : INTEGER                 ; Höhe des Cursors (interner Gebrauch)
  font        : TextFontPtr             ; Zeiger auf aktuellen  Zeichensatz
  algoStyle   : FontStyleSets           ; Zeichenmodi
                                          (kursiv, unterstrichen, ...)
  txFlags     : FontFlagSet             ; textspezifische Flags
  txHeight    : CARDINAL                ; Höhe des aktuellen Zeichensatzes
  txWidth     : CARDINAL                ; durchschnittliche Zeichenbreite
  txBaseline  : CARDINAL                ; Höhe des Zeichensatzes
                                          ohne Unterlängen
  txSpacing   : INTEGER                 ; Zeichenabstand
```

```
  user         : ADDRESS                      ; Zeiger auf Benutzerdaten
                                                (kennen wir schon...)
  longreserved: ARRAY[Ø..1] OF LONGINT        ; reserviert
  wordreserved: ARRAY[Ø..6] OF WORD           ; reserviert
  reserved     : ARRAY[Ø..7] OF BYTE          ; reserviert
END;
```

areaPtrn — Wir werden im Laufe dieses Kapitels noch die AREA-Befehle kennenlernen. Dieses Feld zeigt auf das aktuelle Füllmuster für AREAs.

tmpRas — Einige spezielle Grafikbefehle (vornehmlich die zum Flächenfüllen oder für AREAs) benötigen einen temporären RAM-Bereich, der das gesamte Füllobjekt zwischenspeichern muß. Dieses Feld enthält einen Zeiger auf dessen Record, der an anderer Stelle besprochen werden soll.

areaInfo — Dieses Feld zeigt auf eine spezielle Datenstruktur für AREA-Befehle.

gelsInfo — Hier steht ein Zeiger auf eine spezielle Datenstruktur für die Gels, die wir noch kennenlernen werden.

mask — Hier lassen sich bestimmte Bitplanes ausblenden. Jedes Bit dieses Wertes steht für eine Plane. Ist eines gesetzt, so wird die zugehörige Bitplane auch dargestellt, ist es gelöscht, wird sie ausgeblendet. Normalerweise steht hier der Wert 255, da alle Bitplanes aktiviert sind. Schreiben Sie in dieses Feld aber 0, so ist überhaupt keine Bitplane mehr aktiv.

areaPtSz — Hier steht die Größe für ein selbstdefiniertes Muster. Zulässig sind nur Zweierpotenzen, wobei in diesem Feld nur die Exponenten stehen.

flags — Hier stehen die RastPort-Flags. Nur die Werte der Menge RastPortFlagSet sind zulässig:

- firstDot: Zeichne den ersten Punkt einer Linie?
- oneDot: Benutze den »one dot mode« für Linien, d.h. daß bei einer Linie nur ein Punkt pro Zeile gezeichnet wird, was in den meisten Fällen recht lückenhafte Linien ergibt.
- dBuffer: Double-Buffer-Modus (kommt später noch)

areaOutline: Wird von AREAFILL benutzt.

rpf4: Reserviert

noCrossFill: Wird von AREAFILL benutzt.

rpf6-rpf8: Reserviert

linePtrn

Wer sagt, daß Sie Linien nur durchgezogen darstellen könnten? Nein, auch für sie läßt sich ein Muster definieren. Die Vorgehensweise ist ganz einfach: Sie haben 16 Bits, die Sie nach Belieben setzen oder löschen können. Ein gesetztes Bit repräsentiert einen gesetzten Punkt, ein nicht gesetztes dagegen einen gelöschten. Mit diesem Muster zeichnet GRAPHICS dann die Linie. Als Beispiel sei eine gestrichelte Linie gedacht:

```
Bit 15               Bit 0
        ********
         ........
```

Nun ermitteln Sie den Wert dieses Musters, indem Sie die Werte jedes gesetzten Bits addieren:

2^15+2^13+2^11+2^9+2^7+2^5+2^3+2^1 = 43690

Diesen Wert können Sie nun in das Feld linePtrn des RastPorts schreiben:

MeinWindow^.rPort^.linePtrn = 43690

x

Hier steht die x-Koordinate des sogenannten »Grafik-Cursor«. Der Grafik-Cursor ist als ein unsichtbarer Punkt anzusehen, der dort steht, wo der letzte Punkt einer Zeichenoperation (einer gezogenen Linie, z.B.) gesetzt wurde. Nach dem Öffnen der Zeichenebene befindet sich der Grafikcursor beim Punkt (0;0).

y

In diesem Feld steht die y-Koordinate des eben besprochenen Grafik-Cursors.

font

Dieses Feld enthält einen Zeiger auf die aktuelle Text-Struktur.

algoStyle

Hier stehen die aktuellen Darstellungsmodi eines Zeichensatzes. Die Werte der Menge FontStyleSet sind zulässig:

underlined: Jedes Zeichen wird unterstrichen dargestellt.

bold: Die Fettschrift wird eingeschaltet.

italic: Jedes Zeichen wird kursiv (schräg) auf den Bildschirm gebracht.

extended: Jedes Zeichen wird in doppelter Breite dargestellt.

txFlags In diesem Feld stehen die Flags des aktuellen Fonts, was wir aber eindringlich besprechen werden. Zulässig sind die Einträge der Menge FontFlagSet:

romFont: Zeichensatz im ROM (also TOPAZ 60 oder TOPAZ 80).

diskFont: Zeichensatz auf dem Device fonts:, normalerweise das fonts-Verzeichnis der Startdiskette, kann aber durch den CLI-Befehl ASSIGN beliebig geändert werden.

revPath: Font, der von rechts nach links schreibt

tallDot: Auflösung des Screens hires non-interlaced

wideDot: Auflösung des Screens lores interlaced

proportional: Aktueller Zeichensatz ist ein Proportional-Schriftsatz. Abstand der einzelnen Buchstaben kann variieren.

designed: Nobody knows ...

removed: Der aktuelle Font ist dem System nicht zugänglich, was bei allen noch nicht geöffneten externen Zeichensätzen der Fall ist. Diese Werte sind aber nur für das System interessant.

4.2.2 Der ViewPort-Record

```
ViewPort=RECORD
  next      : ViewPortPtr ; Zeiger auf den nächsten ViewPort
  colorMap  : ColorMapPtr ; Zeiger auf die Farbtabelle des ViewPorts
                            (ColorMap) Falls Nil werden voreingestellte
                            Werte verwendet
  dspIns    : CopListPtr  ; Zeiger auf Copperliste fürs Display
                            (MakeView ())
  sprIns    : CopListPtr  ; Zeiger auf Copperliste für Sprites
  clrIns    : CopListPtr  ; ebenfalls Zeiger auf Copperliste für Sprites
  uCopIns   : UCopListPtr ; Zeiger auf eigene Copperliste
  dWidth    : INTEGER     ; Breite des sichtbaren Bildes (Display)
  dHeight   : INTEGER     ; Höhe des sichtbaren Bildes (Display)
  dxOffset  : INTEGER     ; x-Abstand vom Punkt (Ø;Ø) der Bitmap
  dyOffset  : INTEGER     ; y-Abstand vom Punkt (Ø;Ø) der Bitmap
  modes     : ViewModeSet ; Darstellungsmodi
```

```
  reserved : CARDINAL    ; reserviert
  rasInfo  : RasInfoPtr  ; Zeiger auf RasInfo-Struktur
END;
```

colorMap In diesem Feld steht ein Zeiger auf die Farbtabelle eines Screens. Diese ist wie folgt aufgebaut:

```
ColorMap=RECORD
  flags      : UByte    ; systemreserviert
  type       : UByte    ; Typ
  count      : CARDINAL ; Anzahl der Farben
  colorTable: ADDRESS   ; Adresse der eigentlichen Farbtabelle
END;
```

type Ist dieses Feld gleich 0, was der Normalfall ist, so besteht die eigentliche Farbtabelle aus einer Ansammlung von RGB-Werten vom Typ UWord (CARDINAL bei uns).

dspIns,sprIns,clrIns: Hier stehen die Adressen der System-Copperlisten, was wohl nicht weiter von Interesse sein dürfte.

uCopIns Dieses Feld zeigt auf die aktuelle Benutzer-Copperliste. Eine Benutzer-Copperliste ist eine von Ihnen erstellte Instruktionstabelle für den Copper-Spezialchip. Was der Copper überhaupt ist, und wie Sie Ihre eigene Instruktionsliste definieren, steht in Kapitel 5.

dxOffset Hier steht der x-Abstand zum Punkt (0;0) der Bitmap. Mit diesem und dem nächsten Wert kann der vom ViewPort kontrollierte Screen relativ zum gesamten Bildschirm positioniert werden, wenn Sie z.B. im Preferences-Programm das Bild verschieben, so werden diese Werte geändert. »dxOffset« kann zwischen –16 und +352 bzw. –32 und +704 bei Hires, »dyOffset« zwischen –16 und +256 (+200 bei NTSC) bzw. –32 und +512 (+400 bei NTSC) bei Interlace variieren.

dyOffset In diesem Feld das gleiche wie im Feld »dxOffset«, mit dem Unterschied, daß dieser Wert hier für den y-Anstand gilt.

modes Dieses Feld enthält die Darstellungsmodi des ViewPorts. Jene sind exakt dieselben, wie sie beim Öffnen eines Intuition-Screens im Feld viewModes des NewScreen-Records übergeben werden. Bei Interesse schauen Sie bitte dort nach.

rasInfo Hier steht ein Zeiger auf die RasInfo-Struktur eines ViewPorts. Die RasInfo-Struktur sieht so aus:

```
RasInfo=RECORD
  next    : RasInfoPtr ; Zeiger auf den nächsten RasInfo (für DualPlayfield)
  bitmap  : BitMapPtr  ; Zeiger auf die Bitmap
  rxOffset: INTEGER    ; x-Offset des RastPorts
  ryOffset: INTEGER    ; y-Offset des RastPorts
END;
```

Bitmap zeigt auf den Bitmap-Header, die Kopfstruktur einer Bitmap:

```
BitMap=RECORD
  bytesPerRow: CARDINAL                ; Anzahl der Bytes pro Reihe
  rows       : CARDINAL                ; Anzahl der Reihen
  flags      : UByte                   ; Flags
  depth      : UByte                   ; Anzahl der Bitplanes
  pad        : CARDINAL                ; systemreserviert
  planes     : ARRAY[0..7] OF ADDRESS ; Adressen der einzelnen Planes
END;
```

4.3 Die einfachen Grafikbefehle

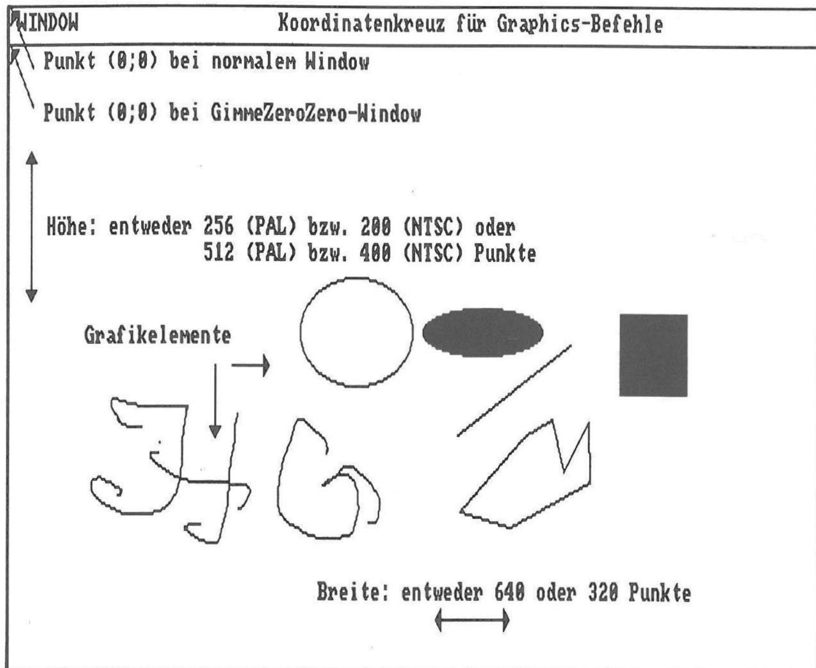


Bild 4.1: Einige Grafikelemente

4.3.1 Kleine Farblehre

Bevor wir nun ans große Zeichnen gehen, ist es vielleicht ganz sinnvoll, sich erst mit den zu verwendenden Farben auseinanderzusetzen. Wie Sie bereits in den vorherigen Kapiteln erfahren haben, steuert die ColorMap alles, was mit Farben zu tun hat. Die Farbwerte beim AMIGA haben grundsätzlich folgendes Format, in dem sie auch in der ColorMap abgelegt werden (hexadezimal):

0RGBH	
R	Rotanteil der Farbe (0–15)
G	Grünanteil der Farbe (0–15)
B	Blauaunteil der Farbe (0–15)

Je nach Anzahl der verwendeten Bitplanes haben wir 2–32 (von den Spezialmodi abgesehen) Farben, die wir zum Zeichnen benutzen können. Um eine neue Farbe zu erstellen, muß man lediglich den Befehl SETRGB4 verwenden:

SetRGB4 (MeinViewPort , Nummer , Rot , Grün , Blau);

MeinViewPort	ViewPortPtr auf den ViewPort, dessen Farben zu setzen sind.
Nummer	CARDINAL-Wert, der die Nummer des zu setzenden Farbregisters angibt.
Rot	CARDINAL-Wert, der den Rotanteil (0–15) der Farbe angibt.
Grün	CARDINAL-Wert, der den Grünanteil (0–15) der Farbe angibt.
Blau	CARDINAL-Wert, der den Blauanteil (0–15) der Farbe angibt.

Dabei sollten Sie beachten, daß hier für die Farbmischung nicht die Gesetze aus dem Kunstunterricht (Wasserfarben – kennen Sie doch), die besagen, daß die drei Grundfarben Rot, Blau und Gelb sind, sondern die physikalischen, die die Farben Rot, Grün und Blau als Grundfarben anerkennen, gelten.

Das direkte Gegenstück zu SETRGB4 ist GETRGB4. Dieser Befehl holt die Farbe aus einem Farbregister und übergibt sie einer ariablen. Die GETRGB4-Funktion wird oft dazu eingesetzt, die augenblicklichen Farbregister zu sichern und sie nach Programmende wieder zu setzen, wenn das Programm z.B. die Farben des Workbench-Screens verändert. Die GETRGB4-Syntax lautet:

Farbe = GetRGB4 (MeineColorMap , Nummer);

MeineColor	MapColorMapPtr auf die ColorMap, aus der eine Farbe geholt werden soll.
Nummer	LONGINT-Wert, der das Register spezifiziert, aus dem der Farbwert gelesen werden soll.
Farbe	LONGCARD-Rückgabewert, der die Farbe im Standardformat enthält.

Es ist aber recht umständlich, 16 oder gar 32 Farben mit SETRGB4 zu setzen, doch auch in diesem Fall existiert eine Abhilfe: LOADRGB4. Mit LOADRGB4 können Sie praktisch eine ganze ColorMap füllen, indem Sie als Parameter einen Zeiger auf eine Tabelle mit Ihren RGB-Farbwerten übergeben. Es ist sehr empfehlenswert, diese in eine INLINE-Datenprozedur einzubinden, aber nun zu der Syntax von LOADRGB4:

LoadRGB4 (MeinViewPort , Farbtabelle , Anzahl);

MeinViewPort	ViewPortPtr auf den ViewPort, dessen Farben gesetzt werden sollen.
Farbtabelle	ADDRESSe der Farbtabelle.
Anzahl	INTEGER-Wert, der die Anzahl der zu setzenden Farben angibt.

Ihre Farben müssen selbstverständlich in der Reihenfolge, in der auch die Farbregister stehen, in Ihrer Tabelle enthalten sein. Also kommt der erste Wert ins erste Register, der zweite ins zweite usw.

Zur Anregung einige Beispielfarben:

0000H	Weiß	06FEH	Himmelblau
0FFFH	Schwarz	06CEH	Hellblau
0F00H	Knallrot	000FH	Blau
0F80H	Rotorange	006DH	Dunkelblau
0F90H	Orange	091FH	Purpur
0FB0H	Goldorange	0C1FH	Violett
0FF0H	Zitronengelb	0FACH	Pink
08E0H	Hellgrün	0C80H	Braun
00F0H	Grün	0A87H	Dunkelbraun
02C0H	Dunkelgrün	0CCCH	Hellgrau
00B1H	Waldgrün	0999H	Mittelgrau
00BBH	Türkis	0666H	Dunkelgrau

So, nun müssen wir nur noch die entsprechenden Farben, mit denen wir zeichnen möchten, aus der Tabelle auswählen und ab geht's ins nächste Kapitel.

Zuerst die Vordergrundfarbe (den sogenannten A-Pen): In ihr wird so ziemlich alles, seien es Linien, Punkte, Rechtecke, Kreise usw., gezeichnet, was im Vordergrund stehen soll. Sie wird mit dem Befehl SETAPEN spezifiziert:

SetAPen (MeinRastPort , Nummer);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, dessen Vordergrundfarbe gesetzt werden soll.
Nummer	CARDINAL-Wert der Farbe, die als Vordergrundstift spezifiziert werden soll.

Neben dem A-Pen gibt es auch den B-Pen, der (fälschlicherweise) gern als Hintergrundfarbe verstanden wird. Dem ist aber nicht so, der Hintergrund wird immer in der Farbe des Registers 0 gezeichnet. Der B-Pen hingegen enthält die Farbe, in der die »Lücken« bei Füllmustern gezeichnet werden, doch das soll uns vorerst nicht interessieren. Die Syntax des SETBPEN-Befehls, der die Farbe des B-Pens setzt, lautet:

SetBPen (MeinRastPort , Nummer);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, dessen Vordergrundfarbe gesetzt werden soll.
Nummer	CARDINAL-Wert der Farbe, die als B-Pen gesetzt werden soll.

Um nachträglich einen ganzen RastPort (und damit ein ganzes Window oder einen ganzen Screen) ohne viel Aufwand einzufärben, ist die Benutzung des SETRAST-Befehls anzuraten. GIMMEZEROZERO-Windows können Sie dann ganz einfach löschen, indem Sie als Farbe die aktuelle Hintergrundfarbe wählen (es sollten GIMMEZEROZERO-Fenster sein, da bei ihnen die Zeichenfläche in einem getrennten RastPort verwaltet wird. Bei normalen Fenstern, wo für Zeichenfläche wie Rahmen und Kopfzeile nur ein RastPort benutzt wird, würden Sie so die beiden letztgenannten mitlöschen, was oft nicht im Sinne des Programmierers sein dürfte, aber mitunter hilfreich sein kann, wenn man Kopfleiste und Fensterrahmen leid ist). Zur Syntax von SETRAST:

SetRast (MeinRastPort , Farbe);

MeinRastPort	RastPortPtr auf den RastPort, der eingefärbt werden soll.
Farbe	CARDINAL-Wert des Farbregister, das verwendet werden soll.

4.3.2 Die Zeichenbefehle

Nun ist es aber an der Zeit, mit den Grafikbefehlen anzufangen. Auch hier gelten die Zeichenmodi (JAM1, JAM2,...), die Sie schon im Intuition-Kapitel kennengelernt haben. Der Grundmodus ist JAM2, was Sie mit dem Befehl SETDRMD ändern können:

SetDrMd (MeinRastPort , Zeichenmodus);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, dessen Zeichenmodus geändert werden soll.
Zeichenmodus	Wert aus der Menge DrawModeSet, der den neuen Zeichenmodus angibt. Für eine nähere Beschreibung der zulässigen Modi siehe Intuition-Kapitel.

Aber jetzt auf zu den »richtigen« Zeichenbefehlen. Fangen wir mit einem Kommando an, das einen einzigen Bildpunkt (Pixel, abgeleitet von »picture element«) auf die Zeichenfläche zaubert:

ok := WritePixel (MeinRastPort , x , y);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem der Punkt gesetzt werden soll.
x	INTEGER-Wert, der die x-Koordinate des Punktes angibt.
y	INTEGER-Wert, der die y-Koordinate des Punktes angibt.
ok	LONGINT-Rückgabewert, der entweder 0 (Punkt gesetzt) oder –1 (Punkt konnte nicht gesetzt werden, weil z.B. die angegebenen Koordinaten außerhalb des Windows liegen).

Um die Farbe eines bereits existierenden Punktes zu lesen, ist das Kommando READPIXEL zu verwenden:

Farbe := ReadPixel (MeinRastPort , x , y);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem sich der Punkt, dessen Farbe gelesen werden soll, befindet.
x	INTEGER-Wert, der die x-Koordinate des Punktes angibt.
y	INTEGER-Wert, der die y-Koordinate des Punktes angibt.

Farbe	LONGINT-Wert, der das Farbregister mit der Farbe des Punktes enthält. Ist dieser Wert gleich –1, so konnte die Farbe des Punktes nicht ermittelt werden, z.B. falls dessen Koordinaten außerhalb des zulässigen Bereichs liegen.

Der nächste Befehl, den ich vorstellen möchte, dient dazu, den Grafik-Cursor zu bewegen, was gerade beim Linienzeichnen, das wir hiernach besprechen werden, von größter Bedeutung ist. Der Befehl heißt MOVE und hat die Syntax:

Move (MeinRastPort , x , y);

MeinRastPort	RastPortPtr auf den RastPort des Windows oder Screens, dessen Grafik-Cursor verschoben werden soll.
x	INTEGER-x-Koordinate des Punktes, an den der Grafik-Cursor bewegt werden soll.
y	INTEGER-y-Koordinate des Punktes, an den der Grafik-Cursor bewegt werden soll.

Nun zum Linien-Befehl: Es ist wichtig, den Grafik-Cursor vorher mit MOVE auf den Startpunkt der Linie zu bewegen. Dann kann die Linie mit DRAWLINE zum Endpunkt gezogen werden:

DrawLine (MeinRastPort , x , y);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem die Linie gezeichnet werden soll.
x	INTEGER-x-Koordinate des Endpunktes.
y	INTEGER-y-Koordinate des Endpunktes.

Verändern Sie das Linienmuster der RastPort-Struktur, zeichnen Sie eine Linie und sehen Sie, was passiert. Ausgefüllte Rechtecke kann man mit RECTFILL erzeugen:

RectFill (MeinRastPort , x1 , y1 , x2 , y2);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem das Rechteck gezeichnet werden soll.
x1	INTEGER-x-Koordinate der linken, oberen Ecke des Rechtecks.
Anzahl	INTEGER-Wert, der die Anzahl der auszugebenden Zeichen enthält.

y1	INTEGER-y-Koordinate der linken, oberen Ecke des Rechtecks.
x2	INTEGER-x-Koordinate der rechten, unteren Ecke des Rechtecks.
y2	INTEGER-y-Koordinate der rechten, unteren Ecke des Rechtecks.

Zu beachten ist, daß das Rechteck stets mit dem aktuellen Füllmuster, was wir einige Seiten später besprechen werden, gefüllt wird. Haben Sie kein solches Muster definiert, wird das Rechteck vollständig ausgefüllt.

Mittlerweile können wir schon viele geometrische Körper erstellen, doch Ellipsen und Kreise (die im Prinzip nur Sonderformen der Ellipsen sind) zu zeichnen, dürfte noch einige Schwierigkeiten mit sich bringen. Doch mit dem Befehl DRAWELLIPSE kann man dem abhelfen:

DrawEllipse (MeinRastPort , xm , ym , rx , ry);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem die Ellipse gezeichnet werden soll.
xm	INTEGER-Wert der x-Koordinate des Ellipsenmittelpunktes.
ym	INTEGER-Wert des y-Koordinate des Ellipsenmittelpunktes.
rx	INTEGER-Wert des x-Radius.
ry	INTEGER-Wert des y-Radius.

Aber trotz dieses Befehls stoßen Sie beim Zeichnen von Kreisen auf Schwierigkeiten – mit der Angabe des gleichen x- und y-Radius ist es leider nicht getan, da die Punkte auf der y-Achse »höher« als die auf der x-Achse sind. Damit sich der Programmierer über diese Schwierigkeiten hinwegsetzen und seine kostbare Zeit für sinnvollere Dinge verwerten kann, hat man das DRAWCIRCLE-Kommando eingeführt, das sich im GfxMacros-Headerfile befindet. Leider hat dieser Befehl einen dicken Haken: korrekte Kreise entstehen nur im Lo-Res-Modus mit 320*256 (bzw. 320*200) Punkten. Möchten Sie einen Kreis in einem der anderen Modi zeichnen, so müssen Sie leider selbst die korrekten Radien für das DRAWELLIPSE-Kommando errechnen, was aber doch schwieriger erscheint, als es ist: Teilen Sie die Zahl der x-Bildpunkte durch die der y-Bildpunkte und dividieren Sie den y-Radius durch diese Zahl.

Nun aber zur Syntax von DRAWCIRCLE:

DrawCircle (MeinRastPort , xm , ym , r);	
MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem der Kreis gezeichnet werden soll.
xm	INTEGER-x-Mittelpunkt des Kreises.
ym	INTEGER-y-Mittelpunkt des Kreises.
r	INTEGER-Radius des Kreises.

Ein weiterer Befehl hat die Aufgabe, beliebige Polygone (Vielecke) zu zeichnen. Als Parameter brauchen nur die Adresse einer Tabelle mit den Koordinaten der Eckpunkte des Polygons und deren Anzahl angegeben werden (der RastPort natürlich auch, aber das ist ja schon beinahe selbstverständlich), schon verbindet GRAPHICS die einzelnen Eckpunkte durch Linien miteinander (das erinnert ein bißchen an Intuition-Borders):

PolyDraw (MeinRastPort , Anzahl , Tabelle);	
MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem das Vieleck gezeichnet werden soll.
Anzahl	INTEGER-Wert, der die Anzahl der Eckepunkte angibt.
Tabelle	ADDRESS-Wert, der die Adresse der Tabelle mit den Eckpunkten enthält, Sie verwenden am besten eine INLINE-Datenprozedur. Der letzte Eckpunkt sollte übrigens dem ersten entsprechen, wie bei den Borders auch.

Beachten Sie bitte, daß Sie vor dem POLYDRAW-Kommando den Grafik-Cursor mit MOVE auf die Stelle des ersten Punktes im Vieleck setzen sollten, sonst erhalten Sie eine Linie zuviel.

Und nun zu einem Befehl für die Textausgabe an der aktuellen Position des Grafik-Cursors, wobei die Buchstaben auch gezeichnet werden, da der AMIGA ja keine Unterscheidung zwischen Text- und Grafikmodus macht, was einen Teil seiner Flexibilität ausmacht. Der Befehl heißt schlicht und ergreifend TEXT und ist eine ernsthafte Alternative zu PRINTITEXT, da seine Handhabung wesentlich einfacher ist:

Text (MeinRastPort , String , Anzahl);	
MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem der Text ausgegeben werden soll.
String	ADDRESSe des auszugebenden Strings.
Anzahl	INTEGER-Wert, der die Anzahl der auszugebenden Zeichen enthält.

Achtung: TEXT schreibt den angegebenen Text nicht UNTER der aktuellen Position des Grafikcursors, sondern darüber!

Bei diesem Befehl gibt es einige Tricks, um die Textausgabe besonders wirkungsvoll zu machen, z.B. Schatten- und Randdruck, die ich hier vorstellen möchte. Sie sollten darauf achten, den Modus JAM1 eingestellt zu haben.

Zum Schattendruck: Dieser einfache, aber nette Effekt wird erreicht, indem man den auszugebenden Text zwei, drei oder mehr Punkte, je nach Tiefe des Schattens, von der Position des normalen Schriftzuges versetzt in schwarzer Farbe zeichnet (es kann auch eine andere Farbe sein, wichtig ist nur, daß sie kontrastreich zur Farbe des eigentlichen Textes und von geringerer Intensität bzw. Helligkeit sein muß).

Und nun der Randdruck: sicher haben Sie ihn schon mal gesehen und gedacht, welch komplizierter, mathematischer Algorithmus doch dahinterstehen muß, aber er ist ganz einfach zu erzeugen. Der Text wird einfach einen Punkt versetzt in alle Himmelsrichtungen gezeichnet, wodurch man ein »Verschmieren« erreicht. Darauf wird an der eigentlichen Position der Text in der Hintergrundfarbe gezeichnet. Fertig.

Bevor wir zum nächsten Kapitel übergehen, sollen noch einige Beispielprogramme, die den Umgang mit den bis jetzt kennengelernten Grafikbefehlen sowie deren Anwendung (als Beispiel habe ich einige mathematische Spielereien verwendet) demonstrieren, und ein Bild, welches die Parameter der Grafikbefehle recht anschaulich erläutert, folgen.

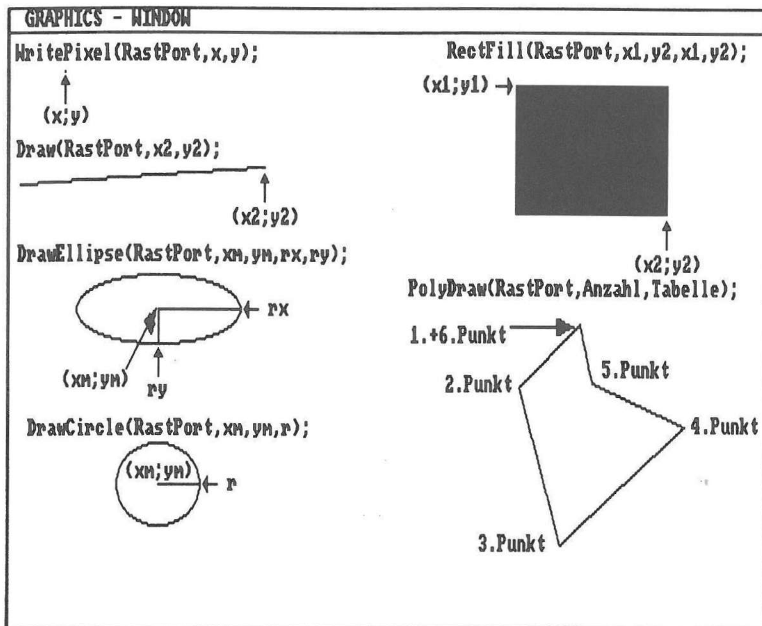


Bild 4.2: Die bis jetzt bekannten Grafikbefehle

```
MODULE GraphicsDemo;

FROM SYSTEM     IMPORT ADR,INLINE;
FROM Strings    IMPORT Length;
FROM Intuition IMPORT NewWindowdow,WindowPtr,OpenWindow,CloseWindow,IDCMPFlags,

IDCMPFlagSet,WindowFlags,ScreenFlags,ScreenFlagSet, WindowFlagSet;
FROM Graphics   IMPORT RastPortPtr,Move,Draw,RectFill,DrawEllipse,Text,GetRGB4,

                       LoadRGB4,PolyDraw,SetAPen,jam1,SetDrMd,SetRast;
FROM GfxMacros IMPORT DrawCircle;
FROM Dos        IMPORT Delay;

VAR MeinWindow : WindowPtr;
    RastPort   : RastPortPtr;
    WindowDaten: NewWindow;
    OldColors  : ARRAY[0..3] OF CARDINAL;
    i          : INTEGER;

PROCEDURE FarbDaten; (* -E- *)
  BEGIN
    INLINE(0000H,0F00H,000FH,0FB0H);
  END FarbDaten;

PROCEDURE PolyDaten; (* -E- *)
  BEGIN
    INLINE(50,50,100,100,200,50,100,250,50,50);
  END PolyDaten;

PROCEDURE TextOutline(x,y:INTEGER; string:ARRAY OF CHAR);
  BEGIN
    Move(RastPort,x+1,y);
    Text(RastPort,ADR(string),Length(string));
    Move(RastPort,x-1,y);
    Text(RastPort,ADR(string),Length(string));
    Move(RastPort,x,y+1);
    Text(RastPort,ADR(string),Length(string));
    Move(RastPort,x,y-1);
    Text(RastPort,ADR(string),Length(string));
    Move(RastPort,x-1,y-1);
    Text(RastPort,ADR(string),Length(string));
    Move(RastPort,x+1,y+1);
    Text(RastPort,ADR(string),Length(string));
    Move(RastPort,x+1,y-1);
    Text(RastPort,ADR(string),Length(string));
    Move(RastPort,x-1,y+1);
    Text(RastPort,ADR(string),Length(string));
```

```
    SetAPen(RastPort,0);
    Move(RastPort,x,y);
    Text(RastPort,ADR(string),Length(string));
  END TextOutline;

BEGIN
  WITH WindowDaten DO
    leftEdge:=0; topEdge:=0; width:=640; height:=256;
detailPen:=0;
    blockPen:=1; idcmpFlags:=IDCMPFlagSet{};
flags:=WindowFlagSet{};
    firstGadget:=NIL; checkMark:=NIL; screen:=NIL; bitMap:=NIL;
    type:=ScreenFlagSet{wbenchScreen}; title:=NIL;
  END;

  MeinWindow:=OpenWindow(WindowDaten);
  RastPort:=MeinWindow^.rPort;

  SetDrMd(RastPort,jam1);

  (* Alte Farben sichern *)
  FOR i:=0 TO 3 DO
OldColors[i]:=GetRGB4(MeinWindow^.wScreen^.viewPort.colorMap,i);
  END;

  SetRast(RastPort,0); (* Alles löschen (auch Titel) *)

  LoadRGB4(ADR(MeinWindow^.wScreen^.viewPort),ADR(FarbDaten),4);

  TextOutline(30,20,"Der Draw-Befehl");
  FOR i:=30 TO 200 BY 3 DO
    SetAPen(RastPort,i MOD 4);
    Move(RastPort,30,30);
    Draw(RastPort,600,i);
  END;
  Delay(200);

  SetRast(RastPort,0);
  SetAPen(RastPort,3);
  TextOutline(30,20,"Die DrawEllipse- und DrawCircle-Befehle");
  FOR i:=10 TO 200 BY 10 DO
    SetAPen(RastPort,i MOD 3);
    DrawEllipse(RastPort,320,100,i,i DIV 4);
  END;
  DrawCircle(RastPort,320,100,i DIV 4);
  Delay(200);
```

```
  SetRast(RastPort,Ø);
  SetAPen(RastPort,3);
  TextOutline(3Ø,2Ø,"Der RectFill-Befehl");
  FOR i:=5Ø TO 2ØØ BY 1Ø DO
    SetAPen(RastPort,i MOD 4);
    RectFill(RastPort,3Ø+(i DIV 1Ø),3Ø+(i DIV 1Ø),i,i);
  END;
  Delay(2ØØ);

  SetRast(RastPort,Ø);
  SetAPen(RastPort,3);
  TextOutline(3Ø,2Ø,"Der PolyDraw-Befehl");
  SetAPen(RastPort,1);
  Move(RastPort,5Ø,5Ø);
  PolyDraw(RastPort,5,ADR(PolyDaten));
  Delay(2ØØ);

  (* Alte Farben wieder setzen *)
  LoadRGB4(ADR(MeinWindow^.wScreen^.viewPort),ADR(OldColors),4);
  CloseWindow(MeinWindow);
END GraphicsDemo.

MODULE Spielerei;

FROM SYSTEM     IMPORT ADR,INLINE;
FROM MathLibØ   IMPORT sin;
FROM Intuition  IMPORT NewWindow,WindowPtr,OpenWindow,CloseWindow,IDCMPFlags,
                       IDCMPFlagSet,WindowFlags,ScreenFlags,ScreenFlagSet,
                       WindowFlagSet;
FROM Graphics   IMPORT RastPortPtr,Move,Draw,DrawEllipse,GetRGB4,
                       LoadRGB4,SetAPen,jam1,SetDrMd,SetRast;
FROM Dos        IMPORT Delay;

VAR MeinWindow : WindowPtr;
    RastPort   : RastPortPtr;
    WindowDaten: NewWindow;
    OldColors  : ARRAY[Ø..3] OF CARDINAL;
    i,a,b,c,d  : INTEGER;

PROCEDURE FarbDaten; (* $E- *)
  BEGIN
    INLINE(ØØØØH,ØFØØH,ØØØFH,ØFBØH);
  END FarbDaten;
```

```
BEGIN
  WITH WindowDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256;
    detailPen:=Ø;
    blockPen:=1; idcmpFlags:=IDCMPFlagSet{};
    flags:=WindowFlagSet{};
    firstGadget:=NIL; checkMark:=NIL; screen:=NIL; bitMap:=NIL;
    title:=NIL;
    type:=ScreenFlagSet{wbenchScreen};
  END;

  MeinWindow:=OpenWindow(WindowDaten);
  RastPort:=MeinWindow^.rPort;

  SetDrMd(RastPort,jam1);

  (* Alte Farben sichern *)
  FOR i:=Ø TO 3 DO

  OldColors±i[:=GetRGB4(MeinWindow^.wScreen^.viewPort.colorMap,i);
  END;

  SetRast(RastPort,Ø); (* Alles löschen (auch Titel) *)

  LoadRGB4(ADR(MeinWindow^.wScreen^.viewPort),ADR(FarbDaten),4);

  FOR i:=Ø TO 639 BY 4 DO
    SetAPen(RastPort,i MOD 3);
    Move(RastPort,Ø,Ø);
    Draw(RastPort,i,TRUNC(4Ø.Ø*sin(REAL(i)/2Ø.Ø)+128.Ø));
  END;
  Delay(2ØØ);

  SetRast(RastPort,Ø);
  FOR i:=Ø TO 66Ø DO
    SetAPen(RastPort,i MOD 3);
    a:=TRUNC(1ØØ.Ø*sin(REAL(i)/1Ø.Ø)+16Ø.Ø);
    b:=TRUNC(6Ø.Ø*sin(REAL(i)/2Ø.Ø)+1ØØ.Ø);
    c:=TRUNC(14Ø.Ø*sin(REAL(i)/3Ø.Ø)+16Ø.Ø);
    d:=TRUNC(8Ø.Ø*sin(REAL(i)/4Ø.Ø)+16Ø.Ø);
    Move(RastPort,a,b);
    Draw(RastPort,c,d);
  END;
  Delay(2ØØ);

  SetRast(RastPort,Ø);
  FOR i:=Ø TO 11Ø BY 2 DO
    SetAPen(RastPort,i MOD 3);
```

```
    DrawEllipse(RastPort,TRUNC(2.Ø*(1.3*REAL(i)+1Ø.Ø)),
      TRUNC(5Ø.Ø*sin(REAL(i)/3Ø.Ø-Ø.2)+REAL(i)+1Ø.Ø),
      TRUNC(2.Ø*(REAL(i)/2.Ø+REAL(i)/8.Ø)), TRUNC(REAL(i)/2.Ø));
  END;
  Delay(2ØØ);

  (* Alte Farben wieder setzen *)
  LoadRGB4(ADR(MeinWindow^.wScreen^.viewPort),ADR(OldColors),4);
  CloseWindow(MeinWindow);
END Spielerei.
```

4.4 Die AREA-Befehle

Die AREA-Befehle sind im Prinzip zum Zeichnen von Polygonen da. Die Unterschiede zu den normalen Zeichenbefehlen und POLYDRAW, der ja auch Polygone zeichnet, sind aber doch recht groß, zumal das definierte Vieleck ausgefüllt wird und durch spezielle Kommandos sehr komplex sein darf (Sie werden erhebliche Schwierigkeiten bekommen, wenn Sie mit POLYDRAW einen Kreis zeichnen möchten). Allerdings ist es für die AREAs vonnöten, gewisse Vorbereitungen zu treffen:

1. Sie müssen mit ALLOCRASTER einen Speicherbereich definieren, der so groß ist, daß er das größte Vieleck fassen kann. Sollten Sie nicht auf total wenig Speicherplatz hin programmieren, wählen Sie am besten die Größe der gesamten Zeichenfläche.
2. Mit INITTMPRAS muß ein temporäres Raster eingerichtet und in den RastPort eingebunden werden.
3. Durch INITAREA müssen Sie dem System zeigen, daß Sie mit AREAs arbeiten möchten und einen Zeiger auf die AreaInfo-Struktur dem RastPort übergeben.

Nach Programmende sollten Sie den allozierten Speicherbereich mit FREERASTER wieder freigeben.

Die verwendeten Befehle haben folgende Syntax:

Speicher := AllocRaster (Breite , Höhe);

Breite	CARDINAL-Wert, der die Breite des Speicherbereichs in Bildpunkten angibt.
Höhe	CARDINAL-Wert, der die Höhe des Speicherbereichs in Bildpunkten angibt.
Speicher	ADDRESSe des breitgestellten Speichers.

InitTmpRas (TempRas , Speicher , Größe);

TempRas — TmpRas-Record (welcher hier nicht beschrieben werden soll, da er unwichtig ist), der initialisiert werden soll.

Speicher — ADDRESS-Wert, der die Adresse des mit ALLOCRASTER allozierten Speicherbereichs angibt.

Größe — LONGINT-Wert, der die Größe des Rasters angibt. Um diese zu ermitteln, ist es ratsam, die Funktion RASSIZE aus GfxMacros anzuwenden, auch wenn sie einen zu großen Wert zurückliefert (siehe Kapitel 6).

Deren Syntax lautet:

```
Größe := RasSize ( Breite , Höhe );
```

Breite: CARDINAL-Wert, der die Breite des Rasters angibt.

Höhe: CARDINAL-Wert, der die Höhe des Rasters angibt.

Größe: CARDINAL-Wert, der die Größe des Rasters enthält.

InitArea (Info , Array , maxKoords);

Info — AreaInfo-Record (kein Zeiger darauf), der initialisiert werden soll (auf ihn kann vom RastPort aus zugegriffen werden, siehe dazu die Beschreibung des RastPorts):

```
AreaInfo=RECORD
  vctrTbl : ADDRESS ; Adresse der Koordinatentabelle
  vctrPtr : ADDRESS ; Adresse des aktuellen Koordinatenpaares
  flagTbl : ADDRESS ; Adresse der Koordinatenflagtabelle
  flagPtr : ADDRESS ; Adresse der Areafill-Flags
  count   : INTEGER ; Anzahl der Koordinatenpaare
  maxCount: INTEGER ; maximale Anzahl von Koordinatenpaaren
  firstX  : INTEGER ; x-Koordinate des ersten Polygon-Punktes
  firstY  : INTEGER ; y-Koordinate des ersten Polygon-Punktes
END;
```

Array — ADDRESSe eines CARDINAL-ARRAYs für spezielle AREA-Informationen. Als Größe des Arrays sollten Sie 250 angeben, das reicht.

maxKoords — INTEGER-Wert, der die maximale Anzahl von Koordinatenpaaren angibt.

FreeRaster (Speicher , Breite , Höhe);

Speicher	ADDRESSe des mit ALLOCRASTER bereitgestellten Speichers
Breite	CARDINAL-Wert, der die Breite des Rasters in Bildpunkten angibt.
Höhe	CARDINAL-Wert, der die Höhe des Rasters in Bildpunkten angibt.

Haben Sie diese Vorbereitungen getroffen, können Sie Ihr Polygon »abstecken«, also die einzelnen Eckpunkte definieren. Mit den Systembefehlen genießen Sie dabei hohen Komfort: Zuerst sollte man das vorhergehende Polygon abschließen und den Startpunkt für das neue angeben. Das ist die Aufgabe von AREAMOVE:

ok := AreaMove (MeinRastPort , x , y);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem ein neuer AREA-Startpunkt definiert werden soll.
x	INTEGER-Wert, der die x-Koordinate des neuen Startpunktes angibt.
y	INTEGER-Wert, der die y-Koordinate des neuen Startpunktes angibt.
ok	LONGINT-Wert, der die Rückmeldung enthält. Verlief alles fehlerfrei, so ist dieser Wert 0, andernfalls -1.

Um einen neuen Eckpunkt in die AREA-Tabelle einzufügen, genügt es, diesen mit AREADRAW zu setzen:

ok := AreaDraw (MeinRastPort , x , y);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem ein neuer AREA-Eckpunkt definiert werden soll.
x	INTEGER-Wert, der die x-Koordinate des neuen Eckpunktes angibt.
y	INTEGER-Wert, der die y-Koordinate des neuen Eckpunktes angibt.

ok	LONGINT-Wert, der die Rückmeldung enthält. Verlief alles fehlerfrei, so ist dieser Wert 0, andernfalls, z.B. wenn nicht genügend Speicher frei war, ist er –1.

Doch auch eine Ellipse kann man an die Polygon-Information anhängen, man braucht nur AREAELLIPSE zu benutzen:

ok := AreaEllipse (MeinRastPort , xm , ym , rx , ry);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, an dessen AREA-Liste die Ellipse angefügt werden soll.
xm	INTEGER-Wert der x-Koordinate des Ellipsenmittelpunktes.
ym	INTEGER-Wert des y-Koordinate des Ellipsenmittelpunktes.
rx	INTEGER-Wert des x-Radius.
ry	INTEGER-Wert des y-Radius.
ok	LONGINT-Rückgabewert, der anzeigt, ob alles fehlerfrei verlaufen ist (0), oder ob ein Fehler auftrat (–1).

Wie bei den normalen Grafikbefehlen auch, gibt es hier einen Kreisbefehl, AREACIRCLE, bei dem exakt dasselbe Problem auftritt, wie bei DRAWCIRCLE. Sehen Sie sich bitte, wenn Sie es noch nicht getan haben, die zugehörige Befehlsbeschreibung an. Zur Syntax von AREACIRCLE:

ok := AreaCircle (MeinRastPort , xm , ym , r);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, an dessen AREA-Liste ein Kreis angefügt werden soll.
xm	INTEGER-x-Mittelpunkt des Kreises.
ym	INTEGER-y-Mittelpunkt des Kreises.
r	INTEGER-Radius des Kreises.
ok	LONGINT-Rückgabewert, der den Verlauf der Operation enthält. Ist er 0, so verlief alles fehlerfrei, ist er aber –1, so trat irgendein Fehler auf (z.B. zuwenig AREA-Speicher).

Um die Vieleck-Definition abzuschließen und die ausgefüllten Polygone zu zeichnen, ist der Befehl AREAEND zu verwenden:

ok := AreaEnd (MeinRastPort);

MeinRastPort	RastPortPtr auf den RastPort des Windows, in dem die definierten Vielecke gezeichnet werden sollen.
ok	LONGINT-Rückgabewert, der anzeigt, ob alles fehlerfrei verlaufen ist (0), oder ob ein Fehler auftrat (–1).

Sie brauchen übrigens nicht, wie bei POLYDRAW oder DRAWBORDER den Startpunkt als letzten Eckpunkt anzugeben. Das erledigt das System automatisch bei der Verwendung von AREAEND.

Es ist noch zu beachten, daß die Polygone mit dem aktuellen Muster, so fern eines vorhanden ist, in der Farbe des A-Pens ausgefüllt werden. Ist kein Muster definiert, werden sie vollständig ausgemalt.

Möchten Sie Ihr eigenes Füllmuster für AREAs oder RECTFILLs definieren, kein Problem: in GfxMacros steht der Befehl SETAFPEN, der genau diese Funktion übernimmt. Dazu muß u.a. die Adresse eines Datenfeldes übergeben werden. Man hat zwei Dinge zur Auswahl: einen ARRAY oder eine INLINE-Prozedur, wobei ich letzteres für besser halte. Dieses Datenfeld muß genau soviele CARDINAL-Werte breit wie hoch sein. Sie haben pro Word (also pro CARDINAL) 16 Bits für Ihr persönliches Füllmuster zur Verfügung. Beispiel für eine Definition:

```
1111111111111111  ØØØØØØØØØØØØØØØØ
ØØØØØØØØØØØØØØØØ  1111111111111111
```

Nun müssen Sie nur noch die einzelnen Bitwerte in Zahlen umrechnen, für unser Beispiel heißt das:

```
ØFFFFH , ØØØØØH , ØØØØØH , ØFFFFH
```

Aber nun zur Syntax des SETAFPEN-Kommandos:

SetAfPen (MeinRastPort , Muster , Anzahl);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, dessen AREA-Füllmuster gesetzt werden soll.
Muster	ADDRESSe des Datenfeldes mit dem Muster (INLINE-Prozedur oder ARRAY).
Anzahl	CARDINAL-Wert, der die Wurzel der Anzahl der Werte in »Muster« angibt. Da diese Anzahl ja immer quadratisch sein muß, kommt hier immer ein wunderbar glatter Wert hin.

Und nun noch zu einem Befehl, der indirekt auch etwas mit AREAs zu tun hat: FLOOD. FLOOD füllt von den angegebenen Koordinaten an alles aus, bis er auf eine andere Farbe trifft. Auch für FLOOD muß ein temporäres Raster, doch keine AreaInfo-Struktur initialisiert werden. Seine Syntax lautet:

ok := Flood (MeinRastPort , Modus , x , y);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, in dem FLOOD angewendet werden soll.
Modus	LONGCARD-Wert, der den Füllmodus angibt: ist er gleich 1, so wird der gesamte Bereich, der die gleiche Farbe wie der Punkt (x;y) hat, bis zu einer Begrenzung ausgefüllt. Ist Modus hingegen 0, so wird solange ausgefüllt, bis die Funktion auf Punkte mit der Farbe des O-Pens trifft. Diese Farbe kann man mit SETOPEN aus GfxMacros setzen:

SetOPen (MeinRastPort , Farbe);

MeinRastPort	RastPortPtr auf den RastPort des Fensters oder Screens, dessen O-Pen gesetzt werden soll.
Farbe	CARDINAL-Wert, der das Register, dessen Farbe gesetzt werden soll, angibt.
x	INTEGER-x-Wert, der die x-Koordinate des Punktes, ab dem gefüllt werden soll, enthält.
y	INTEGER-y-Wert, der die y-Koordinate des Punktes, ab dem gefüllt werden soll, enthält.

Da wir nun auch mit diesem Thema fertig sind, folgt, wie Sie sich bestimmt schon gedacht haben, ein Beispielprogramm:

```
MODULE AreaDemo;

FROM SYSTEM    IMPORT ADR,ADDRESS,INLINE;
FROM Intuition IMPORT NewWindow,WindowPtr,OpenWindow,CloseWindow,IDCMPFlags,
               IDCMPFlagSet,WindowFlags,ScreenFlags,ScreenFlagSet, WindowFlagSet;

FROM Graphics  IMPORT RastPortPtr,AreaMove,AreaDraw,AreaEnd,InitTmpRas,

InitArea,TmpRas,AllocRaster,SetAPen,jam1,SetDrMd, AreaInfo,FreeRaster,SetRast;
FROM GfxMacros IMPORT RasSize,SetAfPen;
FROM Dos       IMPORT Delay;
```

```
VAR MeinWindow   : WindowPtr;
    RastPort     : RastPortPtr;
    WindowDaten  : NewWindow;
    MeinTmpRas   : TmpRas;
    Memory       : ADDRESS;
    MeinAreaInfo: AreaInfo;
    Buffer       : ARRAY[Ø..249] OF CARDINAL;
    ok           : LONGINT;

PROCEDURE MusterDaten; (* $E- *)
  BEGIN
    INLINE(ØFFFFH,ØØØØØH,ØØØØØH,ØFFFFH);
  END MusterDaten;

BEGIN
  WITH WindowDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256;
    detailPen:=Ø;
    blockPen:=1; idcmpFlags:=IDCMPFlagSet{};
    flags:=WindowFlagSet{};
    firstGadget:=NIL; checkMark:=NIL; screen:=NIL;
    bitMap:=NIL;
    type:=ScreenFlagSet{wbenchScreen}; title:=NIL;
  END;

  MeinWindow:=OpenWindow(WindowDaten);
  RastPort:=MeinWindow^.rPort;

  Memory:=AllocRaster(64Ø,256);                   (* Speicher allozieren *)
  InitArea(MeinAreaInfo,ADR(Buffer),1ØØ);         (* AreaInfo initialisieren *)
  InitTmpRas(MeinTmpRas,Memory,RasSize(64Ø,256)); (* TmpRas initialisieren *)
  RastPort^.tmpRas:=ADR(MeinTmpRas);              (* TmpRas übergeben *)
  RastPort^.areaInfo:=ADR(MeinAreaInfo);          (* AreaInfo übergeben *)

  SetDrMd(RastPort,jam1);
  SetRast(RastPort,Ø);
  SetAPen(RastPort,3);
  SetAfPen(RastPort,ADR(MusterDaten),2);          (* Füllmuster definieren *)

  ok:=AreaMove(RastPort,32Ø,78);
  ok:=AreaDraw(RastPort,16Ø,128);
  ok:=AreaDraw(RastPort,32Ø,178);
  ok:=AreaDraw(RastPort,48Ø,128);
  ok:=AreaEnd(RastPort);

  Delay(25Ø);
```

```
  FreeRaster(Memory,640,256);
  CloseWindow(MeinWindow);
END AreaDemo.
```

4.4.1 Füllmuster – auch in Multicolor

Gut – Sie wissen nun, wie man Füllmuster einfach und komfortabel definiert. Doch leider sind mit dieser Methode nur einfarbige Füllmuster möglich. Sie können sich aber eines kleinen Tricks behelfen: Definieren Sie in Ihrer Datenprozedur Ihr Muster für jede Bitplane, je nachdem, welche Punkte in welcher Farbe erscheinen sollen. Geben Sie dann für den Parameter »Anzahl« des SETAFPEN-Befehls 256 minus Ihren Wert an – somit wird der Multicolor-Modus des Systems eingeschaltet. Hier ein konkretes Beispiel:

1. Bitplane (Weiß):

```
1111111111111111  0000000000000000
0000000000000000  1111111111111111
```

2. Bitplane (Schwarz):

```
0000000000000000  1111111111111111
1111111111111111  0000000000000000
```

Daraus ergibt sich folgende INLINE-Prozedur:

```
INLINE(0FFFFH,00000H,00000H,0FFFFH);
INLINE(00000H,0FFFFH,0FFFFH,00000H);
```

Der Befehl, um dieses Muster zu setzen lautet:

SetAfPen(MeinRastPort,ADR(MeinInline),256-2);

Es ist in diesem Fall auch hilfreich, sich die Erklärung für die Sprite-Definition anzusehen. Probieren Sie dieses Muster aus, indem Sie z.B. einen RECTFILL benutzen!

Nun zur Verdeutlichung noch ein Demoprogramm:

```
MODULE MultiPatDemo;

FROM SYSTEM     IMPORT ADR,ADDRESS,INLINE;
FROM Intuition IMPORT
NewWindow,WindowPtr,OpenWindow,CloseWindow,IDCMPFlags,

IDCMPFlagSet,WindowFlags,ScreenFlags,ScreenFlagSet,
                      WindowFlagSet;
FROM Graphics   IMPORT RastPortPtr,RectFill;
FROM GfxMacros IMPORT SetAfPen;
```

```
FROM Exec       IMPORT AllocMem,FreeMem,MemReqs,MemReqSet;
FROM Dos        IMPORT Delay;

VAR MeinWindow   : WindowPtr;
    RastPort     : RastPortPtr;
    WindowDaten  : NewWindow;

PROCEDURE MusterDaten; (* -E- *)
  BEGIN

INLINE(ØFFFFH,ØCCCCH,ØØØØØH,ØCCCCH,ØØØØØH,ØCCCCH,ØØØØØH,ØCCCCH,ØFFFFH);

INLINE(ØØØØØH,ØØØØØH,ØFFFFH,ØØØØØH,ØFFFFH,ØØØØØH,OFFFFH,ØØØØØH,ØØØØØH);
  END MusterDaten;

BEGIN
  WITH WindowDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256;
    detailPen:=Ø;
    blockPen:=1; idcmpFlags:=IDCMPFlagSet{};
    flags:=WindowFlagSet{};
    firstGadget:=NIL; checkMark:=NIL; screen:=NIL; bitMap:=NIL;

    type:=ScreenFlagSet{wbenchScreen}; title:=NIL;
  END;

  MeinWindow:=OpenWindow(WindowDaten);
  RastPort:=MeinWindow^.rPort;

  SetAfPen(RastPort,ADR(MusterDaten),256-3);
  RectFill(RastPort,16Ø,78,48Ø,15Ø);

  Delay(25Ø);

  CloseWindow(MeinWindow);
END MultiPatDemo.
```

4.4.2 Weitere Grafikbefehle

Es gibt neben den schon bekannten, viel benutzen Grafikbefehlen auch einige weitere, weniger verwendete, die ich Ihnen hier noch präsentieren möchte. Erst einmal zwei Befehle aus GfxMacros, die den Zustand des Displays kontrollieren: ONDISPLAY und OFFDISPLAY. OFFDISPLAY schaltet das gesamte Display (also alle Screens samt Windows) bis zu einem ONDISPLAY-Befehl aus. Das ist nützlich, wenn Sie sehr, sehr langwierige Berechnungen (Raytracing-Algorithmus, Fraktale mit 100 000 und mehr Iterationen, etc.) durchführen, da, wenn ein Computer lange Zeit (über einen halben

Tag) unbenutzt eingeschaltet bleibt, die Gefahr besteht, daß sich das aktuelle Bild in die Phosphorschicht des Monitors einbrennt. Beide Befehle sind parameterlos:

OnDisplay;

OffDisplay;

Daneben gibt es noch ein weiteres Kommando aus GfxMacros: BNDRYOFF, das dazu da ist, die Randüberschreitungskontrolle bei AREAs oder RECTFILL abzuschalten. Diese Kontrolle überwacht, ob bei den beiden genannten Befehlen der vom RastPort verwaltete Zeichenbereich eingehalten wird. Ein Abschalten der Randüberwachungskontrolle, das äußerst selten erwünscht ist, könnte also fatale Folgen haben. Die Syntax des BNDRYOFF-Befehls ist:

BndryOff (MeinRastPort);

MeinRastPort	RastPortPtr auf den RastPort, dessen Kontrolle abgeschaltet werden soll.

Mit einem speziellen Befehl, SETWRMSK, der wohl nur für Zeichenprogramme sinnvoll ist, lassen sich auch Bitplanes vor dem Überschreiben schützen. Als Maske dient ein 8-Bit-Wert (BYTE), dessen 7 unteren Bits den Status der Bitplanes repräsentieren: 0 bedeutet geschützt, 1 frei. Beispiel:

Bitplane 6543210
1110011 = 0EBH

In diesem Fall sind die Planes 2 und 4 vor dem Überschreiben geschützt. Mit diesem Verfahren wurde wahrscheinlich auch die »Fix Background«-Option von DPaint II realisiert. Die Syntax von SETWRMSK lautet:

SetWrMsk (MeinRastPort , Maske);

MeinRastPort	RastPortPtr auf den RastPort, dessen Bitplane-Maske gesetzt werden soll.
Maske	BYTE-Wert, der die Maske definiert.

Jetzt seien noch zwei Befehle besprochen, aus denen sich bei geschickter Benutzung einiges herausholen läßt. Die Rede ist von den Scrollbefehlen SCROLLRASTER und SCROLLVPORT. Als Scrolling (von screen rolling) bezeichnet man das Verschieben eines Bildschirminhaltes oder auch des gesamten Bildschirms. Es gibt, grob gesehen, zwei Arten von Scrolling: Hard-Scrolling und Smooth-Scrolling (oder Soft-Scrolling).

Hard-Scrolling ist, wenn ziemlich grob gescrollt wird, beispielsweise im CLI oder einem Editor, wo der Bildschirminhalt zeichenweise (bzw. zeilenweise) aufrückt. Smooth-Scrolling dagegen ist das punktweise Verschieben des Bildschirminhaltes, wie es bei manchen Spielen vorkommt (obwohl es Games gibt, bei denen man den Eindruck gewinnt, es handele sich um Hard-Scrolling). Aber genug der sowieso schon zu langen Vorrede, ab zur näheren Beschreibung beider Befehle (sie scrollen selbstverständlich »smooth«): SCROLLRASTER scrollt einen von Ihnen angegebenen RastPort-Bereich um die Ihnen angegebene Anzahl Bildpunkte:

ScrollRaster (MeinRastPort , ax , ay , x1 , y1 , x2 , y2);

MeinRastPort	RastPortPtr auf den RastPort, der gescrollt werden soll.
ax	INTEGER-Wert, der die Anzahl der Bildpunkte, um die in x-Richtung gescrollt werden soll, angibt. Ist dieser Parameter positiv, so wird nach rechts, ist er negativ, nach links gescrollt.
ay	INTEGER-Wert, der die Anzahl der Bildpunkte, um die in y-Richtung gescrollt werden soll, angibt. Ist dieser Parameter positiv, so wird nach unten, ist er negativ, nach oben gescrollt.
x1	INTEGER-Zahl, die die x-Koordinate der linken, oberen Ecke des zu scrollenden Bereichs angibt.
y1	INTEGER-Wert, der die y-Koordinate der linken, oberen Ecke des zu scrollenden Bereichs angibt.
x2	INTEGER-Zahl, die die x-Koordinate der rechten, unteren Ecke des Scrollbereichs enthält.
y2	INTEGER-Zahl, die die y-Koordinate der rechten, unteren Ecke des Scrollbereichs enthält.

SCROLLVPORT ist eigentlich nur ein Synonym für MOVESCREEN, denn erstgenannter verschiebt einen ganzen Screen, doch ist trotz Angabe eines x-Wertes nur vertikales Scrolling möglich. SCROLLVPORT wird ein wenig langsamer ausgeführt als MOVESCREEN, vollzieht sich aber dennoch in einer 60stel Sekunde:

ScrollVPort (MeinViewPort , ax , ay);

MeinViewPort	ViewPortPtr auf den zu scrollenden ViewPort.
ax	INTEGER-Wert, der die Anzahl der Punkte, um die in x-Richtung gescrollt werden soll, enthält. Da dies jedoch nicht möglich ist, hat dieser Parameter keine Bedeutung.

ay	INTEGER-Wert, der die Bildpunktzahl, um die in y-Richtung gescrollt werden soll, enthält.

Zum Schluß möchte ich Ihnen, der Vollständigkeit halber, drei absolut selten gebrauchte Befehle vorstellen: Mit VBEAMPOS läßt sich die aktuelle Position des Rasterstrahls, der das Monitorbild aufbaut, abfragen (dieser Wert bezieht seine Angaben direkt aus dem Hardware-Register 4):

Position := VBeamPos;

Position	LONGINT-Rückgabewert, der die aktuelle Position des Rasterstrahls enthält.

WAITTOF läßt das Programm warten, bis der Rasterstrahl den nächsten, darzustellenden Screen erreicht hat. WAITTOF hat keine Parameter:

WaitTOF;

Und zum Schluß sei noch WAITBOVP aufgeführt, der das Programm warten läßt, bis der Rasterstrahl den Anfang des angegebenen Screens erreicht hat:

WaitBOVP (MeinViewPort);

MeinViewPort	ViewPortPtr auf den ViewPort des Screens, dessen Anfang erwartet wird.

4.5 Die Spezialmodi – HAM und Halfbrite

Auf dieses Kapitel wurde schon mehrmals verwiesen, aber was sind denn nun die Spezialmodi und was kann man mit ihnen anstellen? 4096 und 64 Farben gleichzeitig darstellen lautet die Antwort. Ja, Sie haben richtig gelesen: mit gewissen Einschränkungen kann man 4096 und 64 Farben gleichzeitig darstellen. Diese Darstellungsarten sind die sogenannten Spezialmodi, die beide durch Setzen einer 6. Bitplane und dem entsprechenden ViewMode-Flag erreicht werden. In der AMIGA-Sprache heißen sie Hold-and-Modify (HAM, Sie werden schon sehen, warum dieser Modus so heißt) bei 4096 und Extra-Halfbrite (EHB) bei 64 Farben. Fangen wir mit dem leichter zu verstehenden Extra-Halfbrite-Modus an:

Wie schon gesagt, wird er dadurch erreicht, daß man einen Screen mit 6 Bitplanes definiert und im Feld viewModes das Flag EXTRAHALFBRITE setzt. Eigentlich müßten wir dann 64 völlig voneinander unabhängige Farben in 64 Registern zu Ver-

fügung haben, was stimmt, bis auf die Tatsache, daß sie nicht unabhängig voneinander sind. EHB erlaubt zwar 64 Farbregister, doch die oberen (32–63) enthalten immer die Farben der unteren 32 (also 0–31) mit der halben Helligkeit, was unsere Freiheit erheblich einschränkt. Technisch wird das dadurch erreicht, daß alle Rot-, Grün- und Blaukomponenten ein Bit nach rechts verschoben werden. So können Sie sich denken, daß, wenn die Grundfarben schon sehr dunkel sind, sich die EHB-Farben kaum davon unterscheiden.

Sollte man es aber geschafft haben, sich damit abzufinden, offenbaren sich recht interessante Effekte für den Programmierer. Gerade wo feine Farbverläufe wichtig sind, bei Malprogrammen oder Raytracern (fotorealistische Bilder nach dem komplizierten Strahlenverfolgungs-Verfahren) ist dieser Modus von Nutzen. EHB funktioniert nur im Lores-Modus (320 Bildpunkte auf der Horizontalen). Beispielprogramm:

```
MODULE HalfbriteDemo;

FROM SYSTEM     IMPORT ADR,INLINE,LONGSET;
FROM Intuition IMPORT NewScreen,ScreenPtr,OpenScreen,CloseScreen,customScreen,
                      NewWindow,WindowPtr,OpenWindow,CloseWindow,WindowFlags,

WindowFlagSet,IDCMPFlags,IDCMPFlagSet,ShowTitle;
FROM Graphics   IMPORT ViewModes,ViewModeSet,RastPortPtr,RectFill,LoadRGB4,
                      SetAPen;
FROM Exec       IMPORT SetSignal;

VAR MeinScreen : ScreenPtr;
    MeinWindow : WindowPtr;
    ScreenDaten: NewScreen;
    WindowDaten: NewWindow;
    MeinRPort  : RastPortPtr;
    i          : INTEGER;

PROCEDURE Farben; (* -E- *)
  BEGIN

    INLINE(0000H,0F00H,0F80H,0F90H,0FD0H,0FF0H,0BF0H,08E0H,00F0H,02C0H);

    INLINE(00B1H,00BBH,00DBH,01FBH,06FEH,06CEH,000FH,061FH,006DH,091FH);

    INLINE(0C1FH,0F1FH,0FACH,0DB9H,0C80H,0A87H,0FFFH,0CCCH,0999H,0666H);
    INLINE(0333H,0111H);
  END Farben;
```

```
BEGIN
  WITH ScreenDaten DO (* Daten initialisieren *)
    leftEdge:=0; topEdge:=0; width:=320; height:=256;
    depth:=6; detailPen:=0; blockPen:=1;
    viewModes:=ViewModeSet{extraHalfbrite};
    type:=customScreen; font:=NIL; defaultTitle:=NIL;
    gadgets:=NIL; customBitMap:=NIL;
  END;

  WITH WindowDaten DO
    leftEdge:=0; topEdge:=0; width:=320; height:=256;
    detailPen:=0;
    blockPen:=1; idcmpFlags:=IDCMPFlagSet{mouseButtons};
    firstGadget:=NIL;
    flags:=WindowFlagSet{activate,backDrop,borderless}; checkMark:=NIL;
    title:=NIL; screen:=NIL; bitMap:=NIL; minWidth:=320; minHeight:=256;
    maxWidth:=320; maxHeight:=256; type:=customScreen;
  END;

  MeinScreen:=OpenScreen(ScreenDaten);
  IF MeinScreen#NIL THEN
    ShowTitle(MeinScreen,FALSE);
    WindowDaten.screen:=MeinScreen;
    MeinWindow:=OpenWindow(WindowDaten);
    IF MeinWindow#NIL THEN
      MeinRPort:=MeinWindow^.rPort;
      LoadRGB4(ADR(MeinScreen^.viewPort),ADR(Farben),32);

      FOR i:=0 TO 31 DO
        SetAPen(MeinRPort,i);
        RectFill(MeinRPort,10*i,0,(10*i)+10,128);
        SetAPen(MeinRPort,i+32);
        RectFill(MeinRPort,10*i,128,(10*i)+10,256);
      END;

      WHILE NOT(MeinWindow^.userPort^.sigBit IN
      SetSignal(LONGSET{},LONGSET{}))
      DO END;

      CloseWindow(MeinWindow);
      END;
    CloseScreen(MeinScreen);
  END;
END HalfbriteDemo.
```

Um einiges komplizierter, wenn auch wesentlich vielseitiger ist der HAM-Modus (wird auch nicht umsonst HAMmer genannt). Auch er wird durch das Setzen eines zusätz-

lichen viewMode-Flags, HAM, und die Definition einer 6. Bitplane eingeschaltet. HAM funktioniert, wie EHB auch, nur bei Lores-Screens und ein Pixel kann sich nur an einer Komponente von seinem Nachbarpixel unterscheiden, was aber bei feinen Farbabstufungen nicht so ins Gewicht fallen dürfte.

Wie ist der HAM-Modus nun zu programmieren? Erstmal erfordert er die Definition von 16 (richtig: sechzehn, nicht 32) Grundfarben, aus denen die anderen entstehen und die in den Registern 0–15 festgelegt werden. In den Registern 16–63 stehen die HAM-Farben, die die Farbe ihres linken Nachbarpixels annehmen und diese nur in einer Komponente verändern dürfen. Welche Komponente verändert werden darf, hängt davon ab, welcher HAM-Typ die Farbe ist:

Register	Bemerkung
0-15	16 Grund- oder Echtfarben, die frei definiert werden können
16-31	HAM-Typ 1, Blaukomponente wird verändert
32-47	HAM-Typ 2, Rotkomponente wird verändert
48-63	HAM-Typ 3, Grünkomponente wird verändert

Wie stark eine Farbkomponente verändert wird, hängt davon ab, welche Farbe eines Typs Sie wählen. Wählen Sie die Farbe 31, so übernimmt diese die Rot- und Grünkomponenten ihrer linken Nachbarfarbe und hat eine Blaukomponente von 15, da:

```
Farbe 16 (HAM-Typ 1, Blaukomponente wird verändert) + 15
(neue Blaukomponente) = 31
```

Ein weiteres Beispiel: Möchten Sie eine Farbe übernehmen und die Grünkomponente auf 12 setzen, so müssen Sie die Farbe 60 wählen:

```
Farbe 48 (HAM-Typ 3, Grünkomponente wird verändert) + 12 (neuer Grünwert) = 60
```

Alles klar? Dann kann ja das nächste Beispielprogramm kommen:

```
MODULE HAMDemo;

FROM SYSTEM    IMPORT ADR,INLINE,LONGSET;
FROM Intuition IMPORT
NewScreen,ScreenPtr,OpenScreen,CloseScreen,customScreen,

NewWindow,WindowPtr,OpenWindow,CloseWindow,WindowFlags,

WindowFlagSet,IDCMPFlags,IDCMPFlagSet,ShowTitle;
FROM Graphics  IMPORT ViewModes,ViewModeSet,RastPortPtr,RectFill,LoadRGB4,
                      SetAPen,SetRGB4;
FROM Exec      IMPORT SetSignal;
FROM Dos       IMPORT Delay;
```

```
VAR MeinScreen : ScreenPtr;
    MeinWindow : WindowPtr;
    ScreenDaten: NewScreen;
    WindowDaten: NewWindow;
    MeinRPort  : RastPortPtr;
    i,j        : INTEGER;

CONST step=10;
      x=70;
      y=48;

PROCEDURE Farben; (* $E- *)
  BEGIN
    INLINE(0000H,0F00H,00F0H,0000H,0808H,0F0FH,0F00H,000FH);
  END Farben;

BEGIN
  WITH ScreenDaten DO (* Daten initialisieren *)
    leftEdge:=0; topEdge:=0; width:=320; height:=256;
    depth:=6; detailPen:=0; blockPen:=1;
    viewModes:=ViewModeSet{ham};
    type:=customScreen; font:=NIL; defaultTitle:=NIL;
    gadgets:=NIL; customBitMap:=NIL;
  END;

  WITH WindowDaten DO
    leftEdge:=0; topEdge:=0; width:=320; height:=256; detailPen:=0;
    blockPen:=1; idcmpFlags:=IDCMPFlagSet{mouseButtons}; firstGadget:=NIL;
    flags:=WindowFlagSet{activate,backDrop,borderless}; checkMark:=NIL;
    title:=NIL; screen:=NIL; bitMap:=NIL; minWidth:=320; minHeight:=256;
    maxWidth:=320; maxHeight:=256; type:=customScreen;
  END;

  MeinScreen:=OpenScreen(ScreenDaten);
  IF MeinScreen#NIL THEN
    ShowTitle(MeinScreen,FALSE);
    WindowDaten.screen:=MeinScreen;
    MeinWindow:=OpenWindow(WindowDaten);
    IF MeinWindow#NIL THEN
      MeinRPort:=MeinWindow^.rPort;
      LoadRGB4(ADR(MeinScreen^.viewPort),ADR(Farben),8);

      SetAPen(MeinRPort,3);
      RectFill(MeinRPort,x-1,y-1,x+17*step+1,y+16*step+1);
```

```
      FOR i:=Ø TO 15 DO
        SetAPen(MeinRPort,32+i);
        RectFill(MeinRPort,x,i*step+y,step+x,i*step+step+y);
        FOR j:=Ø TO 15 DO
          SetAPen(MeinRPort,j+16);

          RectFill(MeinRPort,step+j*step+x,i*step+y,2*step+j*step+x,
              i*step+step+y);
        END;
      END;

      FOR j:=Ø TO 15 DO
        SetRGB4(ADR(MeinScreen^.viewPort),3,Ø,j,Ø);
        Delay(2Ø);
      END;

      WHILE NOT(MeinWindow^.userPort^.sigBit IN SetSignal(LONGSET{},LONGSET{}))
      DO
      END;

      CloseWindow(MeinWindow);
      END;
    CloseScreen(MeinScreen);
  END;
END HAMDemo.
```

Übrigens funktioniert der HAM-Modus auch mit 5 Bitplanes, nur läßt er dann entsprechend weniger Farben zu. Das sollten Sie am obigen Beispielprogramm unbedingt ausprobieren!

4.6 Bewegte Objekte und Animationen

Sicher haben Sie schon mal etwas von »Sprites« gehört, die gerade bei Computerspielen häufig vorkommen. Was sind Sprites? Sprites (engl. »Kobold«) kann man zur großen Gruppe der bewegten Objekte zählen, die frei und ungebunden über den Bildschirm hin- und hersausen können. Es gibt folgende Arten bewegter Objekte: (Hardware-)Sprites: diese sind wohl die bekanntesten Vertreter ihrer Gruppe und bei fast allen Computern vorhanden. Gerade auf dem C64 gibt es eine nicht endende Anzahl von Tricks für Spritemanipulationen. Von den AMIGA-Sprites können ohne irgendwelche Tricks 8 Stück gleichzeitig dargestellt und kontrolliert werden. Jedes darf maximal 16 Bildpunkte breit, 256 hoch sein und vier Farben enthalten, von denen eine transparent ist. Der Mauszeiger ist übrigens auch ein Hardware-Sprite.

VSprites: VSprites oder virtuelle Sprites sind im Prinzip eine Unterart der Hardware-Sprites, erlauben aber spezielle Abfragen (Kollisionen, etc.) und Feinheiten (jedes hat seine eigenen Farben).

Bobs: Die Blitter-Objekte, auch Software-Sprites genannt, sind eigentlich Teile des Screens oder besser der Bitmap. Der Blitter sorgt durch seine Kopieroperationen dafür, daß sie sich bewegen, was auch sehr schnell ist, aber nicht der Geschwindigkeit der Hardware-Sprites standhalten kann. Da sie in den Screen gezeichnete Objekte sind, unterliegen sie auch dessen Bedingungen.

AnimObjects: AnimObjects sind mehrere Bobs, die in Folge gezeigt werden und so den Eindruck einer Bewegung vermitteln. Alle Objekte bis auf die Bobs sind übrigens nur im Lores-Modus möglich. Fangen wir mit den Hardware-Sprites an.

4.6.1 Hardware-Sprites

Wie bereits gesagt, können ohne irgendwelche Tricks maximal acht Hardware-Sprites, die maximal 16 Punkte breit, 256 Punkte hoch und vierfarbig sein müssen, gleichzeitig dargestellt werden. Folgende Farbkombinationen sind möglich, wobei sich immer zwei Sprites drei Farbregister teilen:

```
Sprite   Farbregister
Ø und 1  16 bis 19
2 und 3  2Ø bis 23
4 und 5  24 bis 27
6 und 7  28 bis 31
```

(Die Register 16, 20, 24 und 28 haben keinen Einfluß auf das Aussehen des Sprites, da sie transparent, also durchsichtig, sein müssen.) Diese Farbregister gelten auch, wenn der betreffende Screen weniger als 32 Farben darstellen kann.

Für alle 8 Sprites sind, wenn sie sich überlappen, bestimmte Prioritäten gesetzt, die nicht geändert werden können und somit akzeptiert werden müssen: Sprite 0 ist stets vor allen anderen zu sehen, Sprite 1 ist hinter Sprite 0, Sprite 2 ist hinter Sprite 1, ..., Sprite 7 ist immer das hinterste.

Um dem System klarzumachen, daß Sprites dargestellt werden sollen, muß das Flag SPRITES im Feld viewModes des NewScreen-Records gesetzt werden. Nun müssen die Sprites noch definiert werden, was mit dem Ausfüllen von einem Record geschieht:

```
SimpleSprite=RECORD
  posctldata: ADDRESS  ; Adresse der SpriteImage-Datenstruktur
  height    : CARDINAL ; Höhe des Sprites
  x         : CARDINAL ; x-Koordinate der Sprite-Position beim Einschalten
  y         : CARDINAL ; y-Koordinate der Sprite-Position beim Einschalten
  num       : CARDINAL ; Spritenummer (wird vom System belegt)
END;
```

Posctldata gibt noch ein Rätsel auf; der aufmerksame Leser dürfte sich fragen, was wohl dahinterstecke. Die Antwort ist, daß hier die Adresse einer Datenstruktur mit folgendem Aufbau steht:

```
SpriteImage=RECORD
  posctl  : ARRAY[Ø..1] OF CARDINAL              ; Positionskontrolle
                                                   (noch unwichtig)
  data    : ARRAY[1..height][Ø..1] OF CARDINAL ; eigentliche Sprite-Daten
  reserved: ARRAY[Ø..1] OF CARDINAL              ; reserviert für zukünftige
                                                   Versionen
END;
```

Das Feld data muß die eigentlichen Spritedaten in einer ganz bestimmten Form enthalten, die hier erklärt werden soll: Da Hardware-Sprites immer vierfarbig (im Grunde genommen dreifarbig) sein müssen, sind mindestens zwei Farbinformationen nötig, um 4 Farben zu definieren (wie bei den Bitplanes):

Farbe	Bits	Bemerkung
Ø	ØØ	Transparent
1	1Ø	Punkt in 1. »Schicht« gesetzt
2	Ø1	Punkt in 2. »Schicht« gesetzt
3	11	Punkt in beiden »Schichten« gesetzt

Ein gesetztes Bit repräsentiert einen gesetzten Punkt, ein nicht gesetztes hingegen einen gelöschten. Die erste Dimension des Arrays enthält also die Daten für die erste »Bitplane« (Schicht), die zweite die der zweiten.

Die Daten werden wie bei den Images zu ganzen Hexadezimalzahlen zusammengefaßt und können in eine INLINE-Datenprozedur geschrieben werden. Aber Vorsicht: Bei Amigas mit mehr als 512 Kbyte RAM müssen die Spritedaten (wie die Imagedaten) im CHIP-RAM stehen. Um das zu erreichen, können wir hier unser kleines Speicherkopierprogramm aus dem Intuition-Kapitel gut gebrauchen.

Für die Definition der Sprites sehen Sie sich doch einmal Bild 4.3 (siehe Seite 172) an.

4.6.1.1 Die Hardware-Sprite-Befehle

Jetzt, wo Sie wissen, wie Hardware-Sprites initialisiert werden, möchten Sie sie natürlich auch handhaben. Dazu stellt die GRAPHICS-Bibliothek einige Befehle zur Verfügung.

Der erste Befehl, der besprochen werden soll, ist GETSPRITE. GETSPRITE reserviert für das angegebene Sprite Speicherplatz:

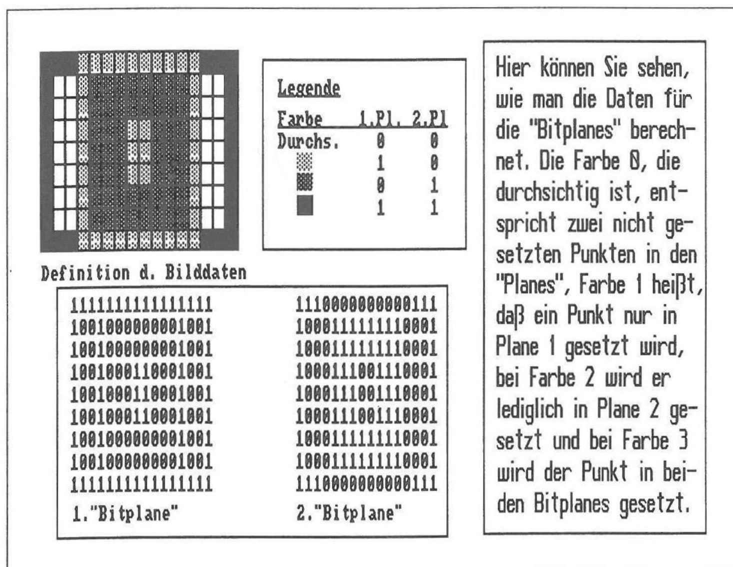


Bild 4.3: Definition eines Hardware-Sprites

Return = GetSprite (MeinSprite , Spritenummer);

MeinSprite	SimpleSpritePtr auf den SimpleSprite-Record des Sprites, das reserviert werden soll.
Spritenummer	INTEGER-Wert, der die Nummer des Sprites angibt. Dieser Wert muß zwischen –1 und 7 liegen, wobei 0 bis 7 die Nummern der Sprites sind. –1 ist dann anzugeben, wenn das nächste freie Sprite reserviert werden soll.
Return	INTEGER-Wert, der die endgültige Nummer des Sprites enthält. Ist dieser Wert –1, dann sind alle Sprite-Controller belegt und kein Speicherplatz konnte für das Sprite reserviert werden. Dieser Wert wird auch in das Feld num der SimpleSprite-Struktur eingetragen.

Achtung: Das Sprite wird nach diesem Befehl noch nicht dargestellt. Es muß erst mit MOVESPRITE positioniert werden.

Natürlich möchte man Sprites auch bewegen (wozu sind Hardware-Sprites auch sonst da?). Auch hierfür gibt es einen Befehl, MOVESPRITE. Sie sollten beachten, daß ein Sprite, das mit GETSPRITE »startklar« gemacht wurde, bis zum ersten MOVESPRITE unsichtbar bleibt. Doch hier die Syntax des MOVESPRITE-Kommandos:

MoveSprite (MeinViewPort , MeinSprite , x , y);

MeinViewPort	ViewPortPtr auf den ViewPort, der das zu bewegende Sprite verwaltet.
MeinSprite	SimpleSpritePtr auf das zu bewegende Sprite.
x	INTEGER-x-Koordinate der neuen Spriteposition.
y	INTEGER-y-Koordinate der neuen Spriteposition.

Um einem Sprite ein neues Aussehen zu geben, kann man mit CHANGESPRITE seine Datenstruktur ändern:

ChangeSprite (MeinViewPort , MeinSprite , NeueDaten);

MeinViewPort	ViewPortPtr auf den ViewPort, der das zu ändernde Sprite verwaltet.
MeinSprite	SimpleSpritePtr auf das zu ändernde Sprite.
NeueDaten	ADDRESS-Wert der neuen Daten (nicht etwa eines SimpleSprite-Records). Siehe auch dazu Kapitel 4.6.1, in dem die Spritedefinition erläutert wird.

Möchte man eines Sprites wieder ledig werden, braucht man nur dessen Nummer dem FREESPRITE-Befehl zu übergeben:

FreeSprite (Nummer);

Nummer	INTEGER-Wert der Nummer des Sprites, das freigegeben wird.

Beachten Sie bitte, daß das Sprite, dessen Nummer an FREESPRITE übergeben wird, nicht nur vom Bildschirm verschwindet, sondern auch aus der internen Liste. Ein mit FREESPRITE freigegebenes Sprite muß vor neuer Benutzung erst mit GETSPRITE wieder initialisiert werden.

Auch GfxMacros steuert mit zwei Befehlen einen Teil zur komfortablen Sprite-Handhabung bei: Das erste Kommando ist OFFSPRITE, es macht alle auf dem Bildschirm befindlichen Sprites unsichtbar, gibt aber nicht deren Speicherplatz frei:

OffSprite;

Als zweites sei ONSPRITE genannt, das die mit OFFSPRITE unsichtbar gemachten Sprites wieder sichtbar macht:

OnSprite;

4.6.1.2 Hardware-Sprites mit 15 Farben

Geben Sie im Feld posctl[1] des SpriteImage-Records, den wir oben definierten, 080H (dezimal 128, das Flag SPRITEATTACHED) an, so werden zwei Sprites, die sich an mindestens einem Punkt überlappen müssen, die Farbregister 17–31 zur Verfügung gestellt. Die 15 Farben stehen nur für den überlappten Bereich zur Verfügung, wobei zu beachten ist, daß das Array des Sprites mit der ungeraden Nummer die höherwertigen und das des Sprites mit der geraden Nummer die niederwertigen Bits darstellen. Zulässig für die »Paarung« sind folgende Spritekombinationen:

0/1, 2/3, 4/5, 6/7

4.6.1.3 Kollisionen mit Hardware-Sprites

Die einfache Handhabung der Hardware-Sprites muß u.a. mit einer umständlichen Kollisionsabfrage erkauft werden. An und für sich haben Sie zwei Möglichkeiten, Kollisionen von Hardware-Sprites zu registrieren:

1. Sie kontrollieren laufend die Position eines Sprites, indem Sie diese in Variablen zwischenspeichern. Erreicht ein Sprite eine bestimmte Schlüsselposition, wird eine Aktion ausgelöst.

2. Sie kontrollieren mit READPIXEL die Umgebung des Sprites und warten auf das Erreichen eines bestimmten Pixels. Diese Methode ist wesentlich schwieriger zu realisieren als die erste.

3. Aber es gibt noch einen dritten Weg, und der führt über die Hardware-Register, die im Kapitel über den Copper recht ausführlich besprochen werden (schließlich sind sie ein wesentlicher Teil der Copper-Programmierung). Nun denn, es gibt im Hardware-Definitionsmodul einen Record namens Custom, in dem die Felder clxcon und clxdat enthalten sind. Diese steuern die Kollisionsabfrage des Systems. Im Feld clxcon können Sie diverse kollisionsspezifische Einstellungen setzen, in clxdat steht der augenblickliche Kollisionsstatus (gesetztes Bit = Zusammenstoß). Die Felder sind wie folgt belegt:

clxcon:

Bits 15–12: Durch Setzen eines dieser Bits können Sie die Kollisionskontrolle für die Sprites mit ungeraden Nummern setzen, sonst werden nur die mit geraden kontrolliert. Das Bit 15 steht für Sprite 7, Bit 12 für Sprite 0. Allerdings können Sie nicht kontrollieren, ob ein Sprite mit gerader oder ungerader

Nummer mit einem Objekt zusammenstieß, es sei denn, die Sprites sind miteinander zu 15-Farb-Sprites verbunden.

Bits 11–6: Hier können Sie bestimmen, welche Bitplanes (6–1) auf Kollision getestet werden sollen.

Bits 5–0: Mit diesen Bits können Sie einstellen, ob eine Berührung mit einem gesetzten oder nicht gesetzten Punkt einer Bitplane (6–1) als Kollision gemeldet werden soll.

clxdat:

Bit 15: unbenutzt
Bit 14: Sprite 4 oder 5 kollidierte mit Sprite 6 oder 7.
Bit 13: Sprite 2 oder 3 kollidierte mit Sprite 6 oder 7.
Bit 12: Sprite 2 oder 3 kollidierte mit Sprite 4 oder 5.
Bit 11: Sprite 0 oder 1 kollidierte mit Sprite 6 oder 7.
Bit 10: Sprite 0 oder 5 kollidierte mit Sprite 4 oder 5.
Bit 9: Sprite 0 oder 1 kollidierte mit Sprite 2 oder 3.
Bit 8: Playfield 2 kollidierte mit Sprite 6 oder 7.
Bit 7: Playfield 2 kollidierte mit Sprite 4 oder 5.
Bit 6: Playfield 2 kollidierte mit Sprite 2 oder 3.
Bit 5: Playfield 2 kollidierte mit Sprite 0 oder 1.
Bit 4: Playfield 1 kollidierte mit Sprite 6 oder 7.
Bit 3: Playfield 1 kollidierte mit Sprite 4 oder 5.
Bit 2: Playfield 1 kollidierte mit Sprite 2 oder 3.
Bit 1: Playfield 1 kollidierte mit Sprite 0 oder 1.
Bit 0: Playfield 1 kollidierte mit Playfield 2.

Aber da für Kollisionsabfragen sowieso kein Mensch Hardware-Sprites nimmt, sei dies nur der Vollständigkeit halber erwähnt.

Nun ist es wohl an der Zeit für ein weiteres Beispielprogramm. Drücken Sie auf die linke Maustaste, folgt ein »Smilie«-Sprite dem Mauszeiger, wohin er sich auch bewegt. Beenden können Sie das Programm durch Anklicken des Schließsymbols:

```
MODULE SpriteDemo;

FROM SYSTEM    IMPORT ADR,ADDRESS,INLINE,LONGSET;
FROM Intuition IMPORT NewWindow,WindowPtr,OpenWindow,WindowFlagSet,customScreen,
                      CloseWindow,IDCMPFlags,IDCMPFlagSet,IntuiMessagePtr,
                      NewScreen,ScreenPtr,OpenScreen,CloseScreen,WindowFlags;
FROM Graphics  IMPORT ViewModes,ViewModeSet,SimpleSprite,ViewPortPtr,SetRGB4,
                      GetSprite,MoveSprite,FreeSprite;
```

```
FROM Exec        IMPORT AllocMem,MemReqs,MemReqSet,FreeMem,GetMsg,ReplyMsg,
                        WaitPort;

TYPE CardPtr = POINTER TO CARDINAL;

VAR MeinSprite : SimpleSprite;
    MeinWindow : WindowPtr;
    MeinScreen : ScreenPtr;
    WindowDaten: NewWindow;
    ScreenDaten: NewScreen;
    VPort      : ViewPortPtr;
    ChipSprite : CardPtr;
    MeineMsg   : IntuiMessagePtr;
    MeineClass : IDCMPFlagSet;
    i          : INTEGER;

PROCEDURE ChipCopy(source: CardPtr; VAR dest: CardPtr; size:
LONGCARD);
VAR ChipPtr: CardPtr;
    copied : LONGCARD;
  BEGIN
    ChipPtr:=AllocMem(size,MemReqSet{chip});
    dest:=ChipPtr;
    copied:=0;
    REPEAT
      ChipPtr^:=source^;
      INC(ChipPtr,2); INC(source,2); INC(copied,2);
    UNTIL copied=size;
  END ChipCopy;

PROCEDURE SpriteDaten; (* $E- *)
  BEGIN
    INLINE(00000H,00000H);
    INLINE(00FC0H,00FC0H);
    INLINE(03FF0H,03030H);
    INLINE(07FF8H,04008H);
    INLINE(07FF8H,04008H);
    INLINE(0F33CH,08CC4H);
    INLINE(0FFFCH,08004H);
    INLINE(0FFFCH,08004H);
    INLINE(0FCFCH,08304H);
    INLINE(0FFFCH,08004H);
    INLINE(0FFFCH,09024H);
    INLINE(07FF8H,04848H);
    INLINE(07FF8H,04788H);
    INLINE(03FF0H,03030H);
```

```
    INLINE(ØØFCØH,ØØFCØH);
    INLINE(ØØØØØH,ØØØØØH);
  END SpriteDaten;

BEGIN
  ChipSprite:=NIL;
  ChipCopy(ADR(SpriteDaten),ChipSprite,64);

  WITH MeinSprite DO
    posctldata:=ChipSprite; height:=14; x:=1ØØ; y:=1ØØ; num:=1;
  END;

  WITH ScreenDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=32Ø; height:=256; depth:=5;
    detailPen:=Ø; blockPen:=1; viewModes:=ViewModeSet{sprites};
    type:=customScreen; font:=NIL; defaultTitle:=NIL; gadgets:=NIL;
    customBitMap:=NIL;
  END;

  WITH WindowDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=32Ø; height:=256; detailPen:=1;
    blockPen:=Ø; idcmpFlags:=IDCMPFlagSet{mouseMove,closeWindow};
    flags:=WindowFlagSet{borderless,reportMouse,windowClose}; firstGadget:=NIL;
    title:=NIL; screen:=NIL; bitMap:=NIL; checkMark:=NIL; type:=customScreen;
  END;

  MeinScreen:=OpenScreen(ScreenDaten);
  WindowDaten.screen:=MeinScreen;
  MeinWindow:=OpenWindow(WindowDaten);

  VPort:=ADR(MeinScreen^.viewPort);

  SetRGB4(VPort,20,15,0,0);
  SetRGB4(VPort,21,0,15,0);
  SetRGB4(VPort,22,0,0,15);
  SetRGB4(VPort,23,15,12,0);

  i:=GetSprite(ADR(MeinSprite),2);
  MoveSprite(VPort,ADR(MeinSprite),100,100);

  LOOP
    WaitPort(MeinWindow^.userPort);
    MeineMsg:=GetMsg(MeinWindow^.userPort);
    WHILE MeineMsg # NIL DO
      MeineClass:=MeineMsg^.class;
      ReplyMsg(MeineMsg);
      IF closeWindow IN MeineClass THEN EXIT;
      ELSE MoveSprite(VPort,ADR(MeinSprite),MeinWindow^.mouseX,
```

```
                    MeinWindow^.mouseY);
      END;
      MeineMsg:=GetMsg(MeinWindow^.userPort);
    END;
  END;

  FreeSprite(2);
  CloseWindow(MeinWindow);
  CloseScreen(MeinScreen);
  FreeMem(ChipSprite,64);
END SpriteDemo.
```

4.6.2 VSprites

Nun wird's interessanter: die VSprites kommen. VSprites (gehören wie die Bobs zur Gattung der Gels) sind den Hardware-Sprites in mancher Beziehung recht ähnlich, da sie auch auf diese zurückgreifen, doch bieten sie viel mehr Möglichkeiten:

- Man ist nicht mehr auf 8 Sprites beschränkt.
- Einfachere Kollisionsabfragen sind möglich.
- Jedes VSprite hat seine eigenen Farben (die nichts mit den Screen-Farben zu tun haben müssen und auch nicht von dessen Tiefe abhängig sind).

Aber wie werden VSprites nun erzeugt? Das Prinzip ist recht einfach: Durch geschickte Rasterinterrupt-Programmierung ist es möglich, ein Hardware-Sprite zur Darstellung mehrerer VSprites zu verwenden, wobei zwischen den VSprites eines Hardware-Sprites mindestens eine Bildschirmzeile liegen muß:

```
############
# Sprite 1 #
############
                                            ############
mindestens eine Bildschirmzeile Abstand    # Sprite 2 #
                                            ############

############
# Sprite 1 #
############
```

Sie sehen, VSprites sind eigentlich »Projektionen« von Hardware-Sprites. Die Priorität einer Projektion liegt natürlich unter der des Originals (sprich Hardware-Sprites). Sie sind demnach an die gleichen Bedingungen wie Hardware-Sprites gebunden: höchstens 16 Pixel breit und 256 hoch bei 4 Farben.

Die Definition der Bilddaten ist übrigens gleich geblieben (nur die reinen Daten), die eines VSprites wird mit dem VSprite-Records erledigt:

```
VSprite=RECORD
  nextVSprite: VSpritePtr     ; nächstes VSprite; nur für das System
  prevVSprite: VSpritePtr     ; vorheriges VSprite; nur für das System
  drawPath   : VSpritePtr     ; nur für das System
  clearPath  : VSpritePtr     ; nur für das System
  oldY       : INTEGER        ; vorherige y-Koordinate; nur für das System
  oldX       : INTEGER        ; vorherige x-Koordinate; nur für das System
  flags      : VSpriteFlagSet ; Flags
  y          : INTEGER        ; y-Koordinate der VSprite-Position
  x          : INTEGER        ; x-Koordinate der VSprite-Position
  height     : INTEGER        ; Höhe des VSprites
  width      : INTEGER        ; Breite des VSprites, nur für Bobs
  depth      : INTEGER        ; Bitplanes des VSprites, nur für Bobs
  meMask     : BITSET         ; Kollisionsmaske
  hitMask    : BITSET         ; Kollisionsmaske
  imageData  : ADDRESS        ; Adresse der Spritedaten
  borderLine : ADDRESS        ; für Kollisionen
  collMask   : ADDRESS        ; Kollisionsmaske
  sprColors  : ADDRESS        ; Adresse der Spritefarben
  vsBob      : BobPtr         ; nur für Bobs
  planePick  : BYTE           ; nur für Bobs
  planeOnOff : BYTE           ; nur für Bobs
END;
```

flags Hier können die VSprite-Flags aus der Menge VSpriteFlagSet angegeben werden:

vsprite: Dieses Flag ist immer zu setzen, wenn VSprites verwendet werden. Handelt es sich um ein Bob, muß dieses Flag gelöscht bleiben.

saveBack: Möchten Sie die Daten unter dem VSprite vor dem Überschreiben retten, so sollten Sie dieses Flag setzen.

overlay: (Nur für Bobs) Durch Setzen dieses Flags können Sie verhindern, daß die nicht gesetzten Punkte eines Bobs in den RastPort übertragen werden sondern durchsichtig sind.

mustDraw: Setzen Sie dieses Flag, wenn Ihr VSprite unbedingt dargestellt werden muß.

vsf4–vsf7: Reserviert.

backSaved: Dieses Flag wird von GRAPHICS gesetzt, wenn der Hintergrund eines Bobs gesichert wird.

bobUpdate: Das hier ist ein temporäres Flag, das von GRAPHICS gesetzt wird und für den Rest der Welt unbrauchbar ist.

gelGone: GRAPHICS setzt dieses Flag, wenn sich mindestens ein VSprite (oder Bob) total außerhalb des Screens befindet und kein Teil von ersterem sichtbar ist.

vsOverflow: Dieses Flag wird von GRAPHICS gesetzt, wenn nicht alle VSprites sichtbar sind, da sich zuviele überlappen. Haben Sie das Flag MUSTDRAW gesetzt, wird das betreffende Sprite (das nicht sichtbare) trotzdem gezeichnet.

meMask Hier wird eine Maske für Kollisionen eingetragen. Siehe dazu das entsprechende Kapitel.

hitMask In diesem Feld steht eine Maske für VSprite-Zusammenstöße. Für nähere Informationen sehen Sie bitte im entsprechenden Kapitel nach.

borderLine Hier stehen Informationen für Sprite-Kollisionen. Siehe dazu das entsprechende Kapitel.

collMask Auch hier steht eine Kollisionsmaske. Siehe dazu das entsprechende Kapitel.

sprColors Da jedes VSprite seine eigenen Farben haben kann, muß hier die Adresse der Farbtabelle des entsprechenden VSprites stehen. Diese Tabelle muß drei Farben im Standardformat (RGB) enthalten und kann sowohl ein Array als auch eine INLINE-Prozedur sein. Die hier definierten Farben können völlig andere als die in den Farbregistern definierten sein!

Neben der VSprite-Struktur muß bei der Verwendung von Gels noch ein weiterer Record initialisiert werden (allerdings nur einmal): GelsInfo.

```
GelsInfo=RECORD
  sprRsrvd     : BYTE          ; Info für Hardware-Sprites (s.u.)
  flags        : BYTE          ; nur für das System
  gelHead      : VSpritePtr    ; erstes Gel der Liste
  gelTail      : VSpritePtr    ; letztes Gel der Liste
  nextLine     : ADDRESS       ; Speicher für vertikale Positon
  lastColor    : ADDRESS       ; Speicher für letzte Farbtabelle
  collHandler  : CollTablePtr ; Zeiger auf Kollisionsroutinen
  leftmost     : INTEGER       ; x-Koordinate der linken,
                                 oberen Ecke des Gel-Rechtecks
```

```
  rightmost     : INTEGER       ; x-Koordinate der rechten, unteren Ecke des
                                  Gel-Rechtecks
  topmost       : INTEGER       ; y-Koordinate der linken, oberen Ecke des
                                  Gel-Rechtecks
  bottommost    : INTEGER       ; y-Koordinate der rechten, unteren Ecke des
                                  Gel-Rechtecks
  firstBlissObj: ADDRESS        ; nur fürs System
  lastBlissObj : ADDRESS        ; nur fürs System
END;
```

sprRsrvd Hier geben Sie die Bitmaske der Hardware-Sprites, die das System für VSprites benutzen kann, an. Benutzen Sie in Ihrem Programm keine reinen Hardware-Sprites, setzen Sie bitte alle Bits dieses Wertes, damit das System erkennt, daß es alle acht Sprite-Controller für seine Zwecke benutzen kann. Verwenden Sie jedoch neben VSprites auch Hardware-Sprites, so geben Sie nur die Bits der Sprites, die für VSprites benutzt werden dürfen an. Bit 0 steht für Sprite 0, Bit 1 für Sprite 1, usw.

gelHead Wenn Sie den Befehl INITGELS, der gleich noch besprochen wird, ausführen, schreibt das System in dieses Feld die Adresse des ersten Gels (Bob oder VSprite) der internen Liste.

gelTail Nach dem Befehl INITGELS steht hier die Adresse des letzten Gels (entweder Bob oder VSprite) der internen Liste.

nextLine In dieses Feld müssen Sie einen Zeiger auf 8 CARDINALs freien Speicherplatz (PUBLIC und MEMCLEAR), den Sie mit ALLOCMEM alloziert haben, schreiben.

lastColor Hier müssen Sie einen Zeiger auf eine Tabelle von 8 Zeigern, die auf 8 Farbtabellen zeigen, eintragen. Der Speicherbereich muß mit ALLOCMEM und den Flags PUBLIC und MEMCLEAR alloziert werden.

collHandler Dieses Feld muß einen Zeiger auf eine Tabelle mit den Adressen Ihrer Kollisionsroutinen enthalten. Alles weitere siehe im Kapitel über Kollisionen.

leftMost-bottommost In diesen vier Feldern müssen Sie die Dimensionen des Rechtecks, in dem sich die Gels aufhalten dürfen, eintragen. Überschreitet ein Gel diese Grenzen, wird sofort eine Randkollision gemeldet.

Haben Sie Ihre Records fertig ausgefüllt, müssen Sie die Adresse des GelsInfo-Records dem RastPort übergeben:

MeinRastPort^.gelsInfo:=ADR(MeinGelsInfo);

Ist auch dieser Schritt getan, muß noch die interne VSprite-Liste initialisiert werden:

InitGels (FirstVSprite , LastVSprite , MeinGelsInfo);

FirstVSprite	VSpritePtr auf einen leeren VSprite-Record, der den Anfang der internen Liste darstellt.
LastVSprite	VSpritePtr auf einen leeren VSprite-Record, der das Ende der internen Liste darstellt.
MeinGelsInfo	GelsInfoPtr auf die von Ihnen initialisierte GelsInfo-Struktur.

Nun können Sie Ihre VSprites, eins nach dem anderen, in die interne Liste einfügen:

AddVSprite (MeinVSprite , MeinRastPort);

MeinVSprite	VSpritePtr auf den Record des VSprites, das in die Liste eingefügt werden soll.
MeinRastPort	RastPortPtr auf den RastPort, der die VSprites kontrollieren soll.

Bis jetzt sind die VSprites erst in der Liste, nicht aber auf dem Bildschirm. Um das zu bewerkstelligen, müssen sie erst sortiert werden:

SortGList (MeinRastPort);

MeinRastPort	RastPortPtr auf den RastPort, dessen Gels sortiert werden sollen.

Danach breiten Sie bitte das System für die Darstellung von Gels vor:

DrawGList (MeinRastPort , MeinViewPort);

MeinRastPort	RastPortPtr auf den RastPort, der die Gels kontrollieren soll.
MeinViewPort	ViewPortPtr auf den ViewPort, der die Gels kontrollieren soll.

Nun müssen Sie noch das System auf die modifizierte Copperliste einstellen:

MrgCop (MeinView);

MeinView	ViewPtr auf den View, in den die neue Copperliste eingefügt werden soll. An diesen kommt man mit der Funktion VIEWADDRESS aus Intuition:

View:= ViewAddress();

View	ViewPtr, der die Adresse des zu holenden Views enthält. Dieser View ist als Record angelegt:

```
View=RECORD
  viewPort   : ViewPortPtr ; Zeiger auf den ersten ViewPort
  lofCprList: CprlistPtr  ; fürs System
  shfCprList: CprlistPtr  ; fürs System
  dyOffset   : INTEGER     ; y-Abstand vom Punkt (Ø;Ø) der Bitmap
  dxOffset   : INTEGER     ; x-Abstand vom Punkt (Ø;Ø) der Bitmap
  modes      : ViewModeSet ; Darstellungsmodi des Views
END;
```

Die Wichtigkeit des Views wurde eigentlich noch gar nicht erläutert: er hält das ganze Display in der Hand. Bei der Besprechung des RastPorts und ViewPorts wurde er deshalb nicht besprochen, weil er so gut wie keine wahnsinnig interessanten Informationen birgt, von den Feldern dyOffset und dxOffset einmal abgesehen.

Als letzen Schritt müssen Sie schließlich den View ins System einfügen:

LoadView (MeinView);

MeinView	ViewPtr auf den View, der in das System eingefügt werden soll. An ihn kommt man mit der schon beschriebenen Funktion VIEW-ADDRESS.

Jetzt können Sie Ihre VSprites nach Herzenslust bewegen, indem Sie die Werte der Felder y und x des entsprechenden VSprite-Records verändern.

Achtung: Nach jeder Änderung der VSprite-Struktur müssen Sie die Anweisungen SORTGLIST bis LOADVIEW ausführen!

Möchten Sie Ihrer VSprites wieder ledig werden, sollten Sie den Befehl REMVSPRITE verwenden:

RemVSprite (MeinVSprite);

MeinVSprite	VSpritePtr auf den Record des Sprites, das Sie loswerden möchten.

Bevor wir nun zu den Kollisionen schreiten, ein Beispielprogramm:

```
MODULE VSpriteDemo;

FROM SYSTEM     IMPORT ADR,ADDRESS,BITSET,INLINE;
FROM Intuition  IMPORT NewWindow,WindowPtr,OpenWindow,WindowFlagSet,customScreen,
                       CloseWindow,IDCMPFlags,IDCMPFlagSet,ViewAddress,
                       NewScreen,ScreenPtr,OpenScreen,CloseScreen,WindowFlags;
FROM Graphics   IMPORT ViewModes,ViewModeSet,VSprite,ViewPortPtr,
                       AddVSprite,GelsInfo,RemVSprite,SortGList,InitGels,
                       MrgCop,LoadView,VSpriteFlags,VSpriteFlagSet,CollTable,
                       RastPortPtr,DrawGList;
FROM Exec       IMPORT AllocMem,MemReqs,MemReqSet,FreeMem,GetMsg,ReplyMsg,
                       WaitPort;
FROM Dos        IMPORT Delay;

TYPE CardPtr = POINTER TO CARDINAL;

VAR vs1,vs2,vs3,vs4,vs5,vs6,vs7,vs8,vs9,vs1Ø: VSprite;
    vsFirst,vsLast      : VSprite;
    MeinWindow          : WindowPtr;
    MeinScreen          : ScreenPtr;
    WindowDaten         : NewWindow;
    ScreenDaten         : NewScreen;
    MeinGelsInfo        : GelsInfo;
    VPort               : ViewPortPtr;
    RPort               : RastPortPtr;
    ChipSprite          : CardPtr;
    i                   : INTEGER;
    SprColors           : ARRAY[Ø..9],[Ø..2] OF CARDINAL;

PROCEDURE ChipCopy(source: CardPtr; VAR dest: CardPtr; size:
LONGCARD);
VAR ChipPtr  : CardPtr;
    copied   : LONGCARD;
  BEGIN
    ChipPtr:=AllocMem(size,MemReqSet{chip});
    dest:=ChipPtr;
    copied:=Ø;
    REPEAT
      ChipPtr^:=source^;
      INC(ChipPtr,2); INC(source,2); INC(copied,2);
    UNTIL copied=size;
  END ChipCopy;
```

```
PROCEDURE SpriteDaten; (* $E- *)
  BEGIN
    INLINE(00FC0H,00FC0H);
    INLINE(03FF0H,03030H);
    INLINE(07FF8H,04008H);
    INLINE(07FF8H,04008H);
    INLINE(0F33CH,08CC4H);
    INLINE(0FFFCH,08004H);
    INLINE(0FFFCH,08004H);
    INLINE(0FCFCH,08304H);
    INLINE(0FFFCH,08004H);
    INLINE(0FFFCH,09024H);
    INLINE(07FF8H,04848H);
    INLINE(07FF8H,04788H);
    INLINE(03FF0H,03030H);
    INLINE(00FC0H,00FC0H);
  END SpriteDaten;

BEGIN
  ChipSprite:=NIL;
  ChipCopy(ADR(SpriteDaten),ChipSprite,64);

  SprColors[0][0]:=0000H; SprColors[0][1]:=0888H; SprColors[0][2]:=0FFFH;
  SprColors[1][0]:=0900H; SprColors[1][1]:=0800H; SprColors[1][2]:=0F00H;
  SprColors[2][0]:=0090H; SprColors[2][1]:=0040H; SprColors[2][2]:=00F0H;
  SprColors[3][0]:=0009H; SprColors[3][1]:=0383H; SprColors[3][2]:=0FF0H;
  SprColors[4][0]:=0000H; SprColors[4][1]:=0888H; SprColors[4][2]:=0F8FH;
  SprColors[5][0]:=0000H; SprColors[5][1]:=0768H; SprColors[5][2]:=07F0H;
  SprColors[6][0]:=0000H; SprColors[6][1]:=0848H; SprColors[6][2]:=0F0FH;
  SprColors[7][0]:=0000H; SprColors[7][1]:=0505H; SprColors[7][2]:=045FH;
  SprColors[8][0]:=0000H; SprColors[8][1]:=0800H; SprColors[8][2]:=006FH;
  SprColors[9][0]:=0000H; SprColors[9][1]:=0808H; SprColors[9][2]:=0F30H;

  WITH vs1 DO
    flags:=VSpriteFlagSet{vsprite}; y:=20; x:=20; height:=14; width:=14;
    depth:=1; meMask:=BITSET{}; hitMask:=BITSET{}; imageData:=ChipSprite;
    sprColors:=ADR(SprColors[0]);
  END;

  WITH vs2 DO
    flags:=VSpriteFlagSet{vsprite}; y:=40; x:=40; height:=14; width:=14;
    depth:=1; meMask:=BITSET{}; hitMask:=BITSET{}; imageData:=ChipSprite;
    sprColors:=ADR(SprColors[1]);
  END;
```

```
WITH vs3 DO
  flags:=VSpriteFlagSet{vsprite}; y:=60; x:=60; height:=14; width:=14;
  depth:=1; meMask:=BITSET{}; hitMask:=BITSET{}; imageData:=ChipSprite;
  sprColors:=ADR(SprColors[2]);
END;

WITH vs4 DO
  flags:=VSpriteFlagSet{vsprite}; y:=80; x:=80; height:=14; width:=14;
  depth:=1; meMask:=BITSET{}; hitMask:=BITSET{}; imageData:=ChipSprite;
  sprColors:=ADR(SprColors[3]);
END;

WITH vs5 DO
  flags:=VSpriteFlagSet{vsprite}; y:=100; x:=100; height:=14;
  width:=14;
  depth:=1; meMask:=BITSET{}; hitMask:=BITSET{}; imageData:=ChipSprite;
  sprColors:=ADR(SprColors[4]);
END;

WITH vs6 DO
  flags:=VSpriteFlagSet{vsprite}; y:=120; x:=120; height:=14;
  width:=14;
  depth:=1; meMask:=BITSET{}; hitMask:=BITSET{}; imageData:=ChipSprite;
  sprColors:=ADR(SprColors[5]);
END;

WITH vs7 DO
  flags:=VSpriteFlagSet{vsprite}; y:=140; x:=140; height:=14;
  width:=14;
  depth:=1; meMask:=BITSET{}; hitMask:=BITSET{}; imageData:=ChipSprite;
  sprColors:=ADR(SprColors[6]);
END;

WITH vs8 DO
  flags:=VSpriteFlagSet{vsprite}; y:=160; x:=160; height:=14;
  width:=14;
  depth:=1; meMask:=BITSET{}; hitMask:=BITSET{}; imageData:=ChipSprite;
  sprColors:=ADR(SprColors[7]);
END;

WITH vs9 DO
  flags:=VSpriteFlagSet{vsprite}; y:=180; x:=180; height:=14;
  width:=14;
  depth:=1; meMask:=BITSET{}; hitMask:=BITSET{}; imageData:=ChipSprite;
  sprColors:=ADR(SprColors[8]);
END;
```

```
WITH vs10 DO
  flags:=VSpriteFlagSet{vsprite}; y:=200; x:=200; height:=14;
  width:=14;
  depth:=1; meMask:=BITSET{}; hitMask:=BITSET{}; imageData:=ChipSprite;
  sprColors:=ADR(SprColors[9]);
END;

WITH MeinGelsInfo DO
  sprRsrvd:=-1;
  nextLine:=AllocMem(16,MemReqSet{public,memClear});
  lastColor:=AllocMem(32,MemReqSet{public,memClear});
  collHandler:=AllocMem(SIZE(CollTable),MemReqSet{public,memClear});
  leftmost:=0; rightmost:=320; topmost:=0; bottommost:=256;
END;

WITH ScreenDaten DO
  leftEdge:=0; topEdge:=0; width:=320; height:=256; depth:=5;
  detailPen:=0; blockPen:=1; viewModes:=ViewModeSet{sprites};
  type:=customScreen; font:=NIL; defaultTitle:=NIL; gadgets:=NIL;
  customBitMap:=NIL;
END;

WITH WindowDaten DO
  leftEdge:=0; topEdge:=0; width:=320; height:=256; detailPen:=1;
  blockPen:=0; idcmpFlags:=IDCMPFlagSet{mouseMove,closeWindow};
  flags:=WindowFlagSet{borderless}; firstGadget:=NIL;
  title:=NIL; screen:=NIL; bitMap:=NIL; checkMark:=NIL;
  type:=customScreen;
END;

MeinScreen:=OpenScreen(ScreenDaten);
WindowDaten.screen:=MeinScreen;
MeinWindow:=OpenWindow(WindowDaten);
VPort:=ADR(MeinScreen^.viewPort);
RPort:=MeinWindow^.rPort;
RPort^.gelsInfo:=ADR(MeinGelsInfo);
InitGels(ADR(vsFirst),ADR(vsLast),ADR(MeinGelsInfo));

AddVSprite(ADR(vs1),RPort);
AddVSprite(ADR(vs2),RPort);
AddVSprite(ADR(vs3),RPort);
AddVSprite(ADR(vs4),RPort);
AddVSprite(ADR(vs5),RPort);
AddVSprite(ADR(vs6),RPort);
AddVSprite(ADR(vs7),RPort);
AddVSprite(ADR(vs8),RPort);
```

```
  AddVSprite(ADR(vs9),RPort);
  AddVSprite(ADR(vs1Ø),RPort);

  FOR i:=2Ø TO 22Ø DO
    vs1.x:=i; vs1.y:=i;

    SortGList(RPort);
    DrawGList(RPort,VPort);
    MrgCop(ViewAddress());
    LoadView(ViewAddress());
  END;

  FOR i:=22Ø TO 2Ø BY -1 DO
    vs1.x:=i; vs1.y:=i;

    SortGList(RPort);
    DrawGList(RPort,VPort);
    MrgCop(ViewAddress());
    LoadView(ViewAddress());
  END;

  Delay(2ØØ);

  RemVSprite(ADR(vs1));
  RemVSprite(ADR(vs2));
  RemVSprite(ADR(vs3));
  RemVSprite(ADR(vs4));
  RemVSprite(ADR(vs5));
  RemVSprite(ADR(vs6));
  RemVSprite(ADR(vs7));
  RemVSprite(ADR(vs8));
  RemVSprite(ADR(vs9));
  RemVSprite(ADR(vs1Ø));

  CloseWindow(MeinWindow);
  CloseScreen(MeinScreen);
  FreeMem(ChipSprite,64);
END VSpriteDemo.
```

4.6.2.1 VSprite-Kollisionen

Bis jetzt haben wir nur VSprites auf den Bildschirm gebracht und sie bewegt. Doch gerade die Kollionsabfrage ist eine hervorstechende Eigenschaft der VSprites. Leider ist diese noch etwas komplizierter als bei den Hardware-Sprites, aber Sie haben noch mehr Möglichkeiten.

Die Felder borderLine und collMask des VSprite-Records werden durch den Befehl INITMASKS automatisch ausgefüllt:

InitMasks (MeinVSprite);

MeinVSprite	VSpritePtr auf das VSprite, dessen Kollisionsmaske gesetzt werden soll.

Doch dazu noch eine Bemerkung: Obwohl INITMASKS die Felder collMask und borderLine initialisiert, muß der Programmierer für diesen Befehl genug Speicher allozieren (der Speicher sollte MEMCLEAR und PUBLIC sein). Die Speichergröße ist folgendermaßen zu berechnen:

```
(collMask): Höhe des Sprites * 2
(borderLine): 2
```

Natürlich muß für jedes Sprite ein anderer Speicherbereich übergeben werden.

Sind diese Felder initialisiert, so müssen Sie noch festlegen, welches Sprite mit welchem kollidieren darf (ist gerade für Spiele besonders nützlich), was mit dem Ausfüllen der Felder meMask und hitMask des VSprite-Records erledigt wird. Sie haben dafür 16 Bits frei, jedes für eine bestimmte Kollisionsart. Im Feld meMask können Sie dann den Wert setzen, der das entsprechende Sprite auszeichnet. Stoßen zwei Sprites mit der gleichen meMask zusammen, passiert überhaupt nichts. In das Feld hitMask müssen Sie dann den Wert schreiben, den ein Sprite, dessen Berührung eine Kollision auslösen soll, in der meMask tragen muß. Bei einer Kollision werden dann meMask und hitMask logisch geANDet und die Routine, die bei einer derartigen Kollision aufgerufen werden soll, ermittelt. Das Bit 1 im hitMask-Feld ist übrigens für Zusammenstöße mit dem Rand reserviert. So eine Kollision wird gemeldet, wenn ein VSprite frecherweise die von uns gesetzten Grenzen (leftmost;topmost)-(rightmost;bottommost) überschreitet und das System springt die Routine 0 an. Das hört sich komplizierter an, als es ist. Gehen wir mal von einem Praxisbeispiel aus:

Angenommen, Sie möchten das wohl allseits bekannte Spiel Pac-Man programmieren (tun Sie das bloß nicht, es gibt bessere Spielideen!). Dazu brauchen Sie fünf VSprites, vier für die Geister und eines für den Pac-Man. Die Geister dürfen einander berühren, ohne daß etwas passiert, berühren sie aber den Pac-Man, so ist eines von dessen Leben futsch. Die Kollisionsmasken lauten also:

```
Geist 1-4: meMask : %ØØØØØØØØØØØØØØ1Ø = BITSET{1}
           hitMask: %ØØØØØØØØØØØØØ1ØØ = BITSET{2}

PacMan   : meMask : %ØØØØØØØØØØØØØ1ØØ = BITSET{2}
           hitMask: %ØØØØØØØØØØØØØØ10 = BITSET{1}
```

Daraus ergeben sich folgende Kollisionstypen:

1. Geist kollidiert mit Geist: %10 mit %100 = %0 (keine Kollision wird gemeldet)

2. Geist kollidiert mit Pac-Man: %10 mit %10 = %10 (Kollision Nr. 1 – Routine Nr. 1 wird angesprungen)
3. Pac-Man kollidiert mit Geist (falls er mitten in diesen hineinknallt): %100 mit %100 = %100 (Kollision Nr. 2 – Routine Nr. 2 wird angesprungen)

Na, ist das so kompliziert?

Aber was ist nun mit den Kollisionsroutinen, die oben ein paar Mal erwähnt wurden? Heißt das, daß bei Kollisionen automatisch eigene Routinen angesprungen werden? Ja, das heißt es. Bevor Sie jetzt Ihre wunderbaren Sprite-Explosionsgrafiken zeichnen und sich Gedanken über die Hintergrundstory eines Spiels, das Sie zu programmieren gedenken, machen, lesen Sie erstmal weiter. Wie erfährt man überhaupt, ob eine Kollision stattgefunden hat? Nun, man benutzt den Befehl DOCOLLISION:

DoCollision (MeinRastPort);

MeinRastPort	RastPortPtr auf den RastPort, der auf VSprite-Zusammenstöße untersucht werden soll.

DOCOLLISION fragt nach, ob eine Kollision stattfand und springt gegebenenfalls automatisch die entsprechende Routine an. Komfortabler geht's wohl nicht.

Um unsere eigenen Auswertungsroutinen für die Zusammenstöße zu definieren, genügt der Befehl SETCOLLISION:

SetCollision (Nummer , Routine , MeinGelsInfo);

Nummer	LONGCARD-Wert, der die Nummer der Kollision, bei der die Prozedur »Routine« angesprungen wird, angibt. Wird hier der Wert 0 angegeben, wird »Routine« bei jeder Randüberschreitung angesprungen.
Routine	PROC-Wert, der die Routine, die bei der Kollision Nummer »Nummer« angesprungen wird, enthält.
MeinGelsInfo	GelsInfoPtr auf den GelsInfo-Record Ihrer VSprites.

Wichtig ist noch, daß Ihre Prozedur in jedem Fall zwei Parameter haben muß, die das System ihr übergibt:

Bei Kollisionen mit dem Rand:

– die Struktur des »schuldigen« Sprites, das die Kollision auslöste, vom Typ VSprite

- einen INTEGER-Wert, der bei Randkollisionen den Code des Randes, mit dem das Sprite zusammenstieß, enthält (Konstanten aus Graphics):

borderhit(# 0):	Randkollision
topHit (#1):	Kollision mit oberem Rand
bottomhit(# 2):	Kollision mit unterem Rand
leftHit (# 4):	Kollision mit linkem Rand
rightHit (# 5):	Kollision mit rechtem Rand

Bei Kollisionen mit einem anderen Sprite:

- die Struktur des Sprites, das die Kollision auslöste, vom Typ VSprite
- die Struktur des Sprites, mit dem das auslösende Sprite zusammenstieß (ebenfalls vom Typ VSprite)

Soviel zur Theorie, wie sie das ROM-KERNEL-MANUAL lehrt. Die Praxis sieht in einigen Belangen etwas anders aus: Da Ihre Kollisionsroutine ungücklicherweise als parameterlose Prozedur des Typs PROC definiert wurde, könnten Sie doch eigentlich keine Parameter vom System empfangen, oder? Nun, wer in Modula-2-Typenlehre »sehr gut« steht, wird wissen, daß PROC nichts anderes als die Adresse eines Moduls angibt (ich selbst mußte bei der Firma A+L anrufen, um diese Frage zu klären). So kann man einfach per ADR (MeineProzedur) die Adresse einer parameterbestückten Kollisionsprozedur übergeben. Die Parameter sollten stets so aussehen:

Für Kollisionen mit dem Rand:

```
PROCEDURE RandKollision (Mask: RandSet; VAR vs: VSprite);
```

Für Kollisionen mit einem anderen Gel:

```
PROCEDURE GelKollision (VAR vs1,vs2: VSprite);
```

Zu den Parametern für Rand-Kollisionen ist noch zu sagen, daß die Kollisionsmaske, die im Definitionsmodul Graphics des M2-Compilers verwendet wird, nicht ganz einsichtig ist. Deshalb sollten die M2-Freaks die Definition des Benchmark-Compilers verwenden und für den Parameter »Mask« den Typ RandSet (SET OF [0..31]) benutzen. Definiert man jetzt noch die einzelnen Rand-Kollisionstypen um, hat man es bei der Abfrage viel leichter. Hier die neuen Typen:

```
TYPE RandSet = SET OF [Ø..31];

CONST newTopHit    = Ø;
      newBottomHit = 1;
      newLeftHit   = 2;
      newRightHit  = 3;
```

So können Sie nun ganz einfach feststellen, um welche Rand-Kollision es sich handelt:

```
IF (newTopHit IN Mask) THEN ... (* Kollision mit dem oberen Rand *)
IF (newBottomHit IN Mask) THEN ... (* Kollision mit dem unteren Rand *)
IF (newLeftHit IN Mask) THEN ... (* Kollision mit dem linken Rand *)
IF (newRightHit IN Mask) THEN ... (* Kollision mit dem rechten Rand *)
```

Falls es Sie interessiert: GRAPHICS schreibt die Adresse Ihrer Kollisionsprozedur sofort an die richtige Stelle im Sprungverteiler, auf den collHandler im GelsInfo-Record zeigt.

Nach soviel Vorteilen kommt noch ein Haken (es wäre ja auch zu schön gewesen ...): Diese Methode stellt keine Möglichkeit zur Verfügung, Kollisionen mit dem Hintergrund abzufragen. Sie müssen da auf die READPIXEL-Methode zurückgreifen, die ich schon im Zusammenhang mit Hardware-Sprites erklärte.

Bevor wir zum Demoprogramm gehen, sollte noch gesagt werden, daß ein Programm, welches VSprite-Kollisionen benutzt, (zumindest beim M2-Compiler) mit REMAKE-DISPLAY aus Intuition abgeschlossen werden und keine Funktionen aus DOS beinhalten sollte, da sonst in einigen Fällen das Diskettenbetriebssystem zusammenbricht (die interne Floppy produziert ein nicht aufhörendes Knallen, Knatschen und Krachen). Es kann unter Umständen auch eine Guru-Meditation erzeugen (warum ???), doch um dem abzuhelfen, lassen Sie vor dem Start des kommenden Programms doch das normale VSprite-Demoprogramm laufen.

Nun ein Beispielprogramm:

```
MODULE CollDemo;

FROM SYSTEM    IMPORT ADR,ADDRESS,BITSET,INLINE;
FROM Intuition IMPORT NewWindow,WindowPtr,OpenWindow,WindowFlagSet,customScreen,
                      CloseWindow,IDCMPFlags,IDCMPFlagSet,ViewAddress,NewScreen,
                      RemakeDisplay,ScreenPtr,OpenScreen,CloseScreen,
                      WindowFlags;
FROM Graphics  IMPORT ViewModes,ViewModeSet,VSprite,ViewPortPtr,RastPortPtr,
                      AddVSprite,GelsInfo,RemVSprite,SortGList,InitGels,MrgCop,
                      VSpriteFlags,VSpriteFlagSet,CollTable,InitMasks,DrawGList,
                      SetCollision,DoCollision,LoadView;
FROM Exec      IMPORT AllocMem,MemReqs,MemReqSet,FreeMem;

TYPE CardPtr = POINTER TO CARDINAL;

VAR vs1,vs2,vsFirst,vsLast : VSprite;
    MeinWindow             : WindowPtr;
    MeinScreen             : ScreenPtr;
    WindowDaten            : NewWindow;
    ScreenDaten            : NewScreen;
```

```
    MeinGelsInfo     : GelsInfo;
    VPort            : ViewPortPtr;
    RPort            : RastPortPtr;
    ChipSprite       : CardPtr;
    i                : LONGCARD;
    SprColors        : ARRAY[0..9],[0..2] OF CARDINAL;
    Mask1,Mask2,Line1,Line2: ADDRESS;
    MeinCollTable    : CollTable;

(* Kollisionsprozedur, die aufgerufen wird, wenn vs1 mit vs2 kollidiert,
   was garantiert der Fall sein wird. Diese Prozedur ist parameterlos,
   da nähere Informationen über die Kollision hier unwichtig sind. *)
PROCEDURE SpriteColl;
  BEGIN
    RemVSprite(ADR(vs2));
  END SpriteColl;

PROCEDURE ChipCopy(source: CardPtr; VAR dest: CardPtr; size: LONGCARD);
VAR ChipPtr: CardPtr;
    copied : LONGCARD;
  BEGIN
    ChipPtr:=AllocMem(size,MemReqSet{chip});
    dest:=ChipPtr;
    copied:=0;
    REPEAT
      ChipPtr^ :=source^ ;
      INC(ChipPtr,2); INC(source,2); INC(copied,2);
    UNTIL copied=size;
  END ChipCopy;

PROCEDURE SpriteDaten; (* $E- *)
  BEGIN
    INLINE(00FCOH,00FC0H);
    INLINE(03FF0H,03030H);
    INLINE(07FF8H,04008H);
    INLINE(07FF8H,04008H);
    INLINE(0F33CH,08CC4H);
    INLINE(0FFFCH,08004H);
    INLINE(0FFFCH,08004H);
    INLINE(0FCFCH,08304H);
    INLINE(0FFFCH,08004H);
    INLINE(0FFFCH,09024H);
    INLINE(07FF8H,04848H);
    INLINE(07FF8H,04788H);
    INLINE(03FF0H,03030H);
```

```
    INLINE(00FC0H,00FC0H);
  END SpriteDaten;

BEGIN
  ChipSprite:=NIL;
  ChipCopy(ADR(SpriteDaten),ChipSprite,64);
  Mask1:=AllocMem(28,MemReqSet{chip});
  Line1:=AllocMem(2,MemReqSet{chip});
  Mask2:=AllocMem(28,MemReqSet{chip});
  Line2:=AllocMem(2,MemReqSet{chip});

  SprColors[0][0]:=0000H; SprColors[0][1]:=0888H;
  SprColors[0][2]:=0FFFH;
  SprColors[1][0]:=0900H; SprColors[1][1]:=0800H;
  SprColors[1][2]:=0F00H;

  WITH vs1 DO
    flags:=VSpriteFlagSet{vsprite}; y:=20; x:=20; height:=14; width:=14;
    depth:=1; meMask:=BITSET{1}; hitMask:=BITSET{2};
    imageData:=ChipSprite;
    sprColors:=ADR(SprColors[0]); collMask:=Mask1; borderLine:=Line1;
  END;

  WITH vs2 DO
    flags:=VSpriteFlagSet{vsprite}; y:=40; x:=40; height:=14; width:=14;
    depth:=1; meMask:=BITSET{2}; hitMask:=BITSET{1};
    imageData:=ChipSprite;
    sprColors:=ADR(SprColors[1]); collMask:=Mask2; borderLine:=Line2;
  END;

  WITH MeinGelsInfo DO
    sprRsrvd:=-1;
    nextLine:=AllocMem(16,MemReqSet{public,memClear});
    lastColor:=AllocMem(32,MemReqSet{public,memClear});
    collHandler:=ADR(MeinCollTable);
    leftmost:=0; rightmost:=320; topmost:=0; bottommost:=256;
  END;

  WITH ScreenDaten DO
    leftEdge:=0; topEdge:=0; width:=320; height:=256; depth:=5;
    detailPen:=0; blockPen:=1; viewModes:=ViewModeSet{sprites};
    type:=customScreen; font:=NIL; defaultTitle:=NIL; gadgets:=NIL;
    customBitMap:=NIL;
  END;
```

```
WITH WindowDaten DO
  leftEdge:=0; topEdge:=0; width:=320; height:=256; detailPen:=1;
  blockPen:=0; idcmpFlags:=IDCMPFlagSet{mouseMove,closeWindow};
  flags:=WindowFlagSet{borderless}; firstGadget:=NIL;
  title:=NIL; screen:=NIL; bitMap:=NIL; checkMark:=NIL;
  type:=customScreen;
END;

MeinScreen:=OpenScreen(ScreenDaten);
WindowDaten.screen:=MeinScreen;
MeinWindow:=OpenWindow(WindowDaten);

VPort:=ADR(MeinScreen^.viewPort);
RPort:=MeinWindow^.rPort;

InitGels(ADR(vsFirst),ADR(vsLast),ADR(MeinGelsInfo));
RPort^.gelsInfo:=ADR(MeinGelsInfo);

InitMasks(ADR(vs1));
InitMasks(ADR(vs2));

FOR i:=1 TO 15 DO
  SetCollision(i,SpriteColl,ADR(MeinGelsInfo));
END;

AddVSprite(ADR(vs1),RPort);
AddVSprite(ADR(vs2),RPort);

FOR i:=20 TO 220 DO
  vs1.x:=INTEGER(i); vs1.y:=INTEGER(i);

  SortGList(RPort);
  DrawGList(RPort,VPort);
  MrgCop(ViewAddress());
  LoadView(ViewAddress());
END;

FOR i:=220 TO 20 BY -1 DO
  vs1.x:=INTEGER(i); vs1.y:=INTEGER(i);

  SortGList(RPort);
  DoCollision(RPort); (* Kollision nur auf dem Rückweg registrieren *)
  DrawGList(RPort,VPort);
  MrgCop(ViewAddress());
  LoadView(ViewAddress());
END;

RemVSprite(ADR(vs1)); (* nur Sprite 1 entfernen, da Sprite 2
                         automatisch bei der Kollision verschwindet *)
```

```
  RemakeDisplay;          (* einen schönen Abschluß erzeugen *)

  CloseWindow(MeinWindow);
  CloseScreen(MeinScreen);
  FreeMem(ChipSprite,64); (* belegten Speicher freigeben, der Rest wird
  FreeMem(Mask1,28);         durch Schließen des Screens wieder beschreibbar *)
  FreeMem(Mask2,28);
  FreeMem(Line1,2);
  FreeMem(Line2,2);

END CollDemo.
```

4.7 Bobs

Bobs (Blitter-Objects) sind eigentlich nichts anderes als Teile des Screens (oder vielmehr der Bitmap), die mit dem Blitter-Chip durch die Gegend bewegt werden. Anders als die VSprites, dürfen die Bobs fast beliebig groß sein, nur muß Ihre Breite durch 16 teilbar sein Die Bob-Breite muß im Feld width des VSprite Records in CARDINALs angegeben werden (daher muß sie auch durch 16 teilbar sein). Da Bobs Teile der Bitmap sind, sind sie natürlich auch vom RastPort abhängig und werden in dessen Auflösung und Bitplane-Tiefe dargestellt. Auch die Felder planePick und planeOnOff haben jetzt eine Bedeutung: in planePick steht, welche Bitplane des Bobs in welche der Bitmap hineinkopiert werden soll. Soll die erste Plane Ihres Bobs in die zweite des RastPorts und die zweite des Bobs in die vierte des RastPorts geschrieben werden, ergibt sich folgendes Ergebnis (jedes Bit des BYTEs steht für eine Plane):

%00001010 = 9

Im Feld planeOnOff steht dagegen, was mit den nicht selektierten Bitplanes des RastPorts passieren soll. Steht hier eine »leere Menge«, so passiert mit ihnen überhaupt nichts, steht dort aber z.B. 3 (%00000100), so wird der sogenannte »Image-Shadow«, der gleich noch besprochen wird, in die 3. Bitplane geschrieben.

Vor dem ganzen Struktur-Kram steht selbstverständlich die Definition der eigentlichen Bob-Daten, das Aussehen. Die Vorgehensweise zur Definition eines Bob-Bildes ist weitgehend identisch mit der eines VSprites, aber für jede Bitplane muß eine Bob-Definitionsstruktur vorliegen, damit das System weiß, wie es das Bob in den verschiedenen Planes zu zeichnen hat. Außerdem sollte jedes Bob sein eigenes Image haben, auch wenn diese identisch sind.

Um nun Bobs darzustellen, muß eine GelsInfo- und eine VSprite Struktur erstellt werden, die ja jetzt wohl hinreichend bekannt sind. Bei VSprite-Record ist zu beachten, daß das Flag VSPRITE im Feld flags (natürlich) GELÖSCHT sein muß und daß vsBob auf den Bob-Record des Bobs zeigen muß. In den Feldern width und depth sind dann

Breite und Bitplane-Tiefe des Bobs einzutragen. Zuzüglich dazu ist noch ein spezieller Bob-Record vonnöten:

```
Bob=RECORD
  flags       : BobFlagSet    ; Bob-Flags
  saveBuffer  : ADDRESS       ; Hintergrundspeicher
  imageshadow: ADDRESS        ; Image-Shadow-Speicher
  before      : BobPtr        ; für Animationen
  after       : BobPtr        ; für Animationen
  bobVSprite  : VSpritePtr    ; Zeiger auf VSprite-Struktur des Bobs
  bobComp     : AnimCompPtr   ; nur für Animationen
  dBuffer     : DBufPacketPtr ; für Double-Buffering, kommt gleich noch
END;
```

flags Hier stehen einige bobspezifische Flags aus der Menge BobFlagSet:

saveBob: Durch das Setzen dieses Flags wird erreicht, daß der Hintergrund, über dem sich ein Bob befindet, nicht gesichert wird.

bobIsComp: Dieses Flag wird benutzt, um Animationen zu erzeugen, was wir auch bald noch machen werden. Es dient dazu, ein Bob als Animationsobjekt zu kennzeichnen.

bf2–bf7: Reserviert

bWaiting: GRAPHICS setzt dieses Flag, wenn bei einer Animation das Bob, dessen Zeiger in after steht, gezeichnet wird.

bobDrawn: Dieses Flag wird von GRAPHICS gesetzt, wenn das Bob augenblicklich vom DRAWGLIST-Befehl gezeichnet wird.

bobsAway: GRAPHICS setzt dieses Flag, wenn das Bob entfernt wird und beim nächsten Aufruf vom DRAWGLIST nicht mehr zu sehen sein soll.

bobNix: GRAPHICS setzt dieses Flag, wenn das Bob nun vollständig entfernt ist (dieses Flag folgt BOBSAWAY).

savePreserve: ???

outStep: GRAPHICS setzt dieses Flag, wenn beim Double-Buffer-Modus beide Buffer gelöscht werden.

saveBuffer Haben Sie das Flag SAVEBACK im Feld flags der VSprite-Struktur gesetzt, muß in diesem Feld die Adresse auf einen Speicherbereich stehen, der so groß ist wie das größte und bunteste darzustellende

Bob sein und im Chip-Memory liegen muß. Die Anzahl der benötigten Bytes läßt sich folgendermaßen errechnen:

```
Breite in Words (CARDINALs) * Höhe in Bildpunkten * Tiefe in Bitplanes
                              (Bitplanes+1 bei ImageShadow) *2
```

imageshadow In diesem Feld stehen die gleichen Daten wie im Feld collMask des VSprite-Records, das logische ODER (OR) aller Zeilen des Bobs. Daher muß imageShadow auch genausoviel Chip-Speicher wie collMask zugewiesen werden. Der Inhalt dieses Feldes wird in die mit planePick aus VSprite definierten Bitplanes geschrieben.

Gut, das war wieder ein schöner Brocken an Strukturen. Kommen wir nun zur Darstellung der Bobs. Diese ist fast identisch mit der der VSprites, nur ist zu beachten, daß statt des ADDVSPRITE-Befehls der ADDBOB-Befehl stehen muß:

AddBob (MeinBob , MeinRastPort);

MeinBob BobPtr auf das Bob, das in die Gel-Liste eingefügt werden soll.

MeinRastPort RastPortPtr auf den RastPort, in dessen Gel-Liste das Bob eingefügt werden soll.

Da Bobs vom RastPort abhängig sind, werden sie beim Schließen des Screens automatisch entfernt. Möchten Sie aber ein Bob vor dem Programmende von der Bildfläche verschwinden lassen, so ist es ganz empfehlenswert, dieses mit dem REMIBOB-Kommando zu tun:

RemIBob (MeinBob , MeinRastPort , MeinViewPort);

MeinBob BobPtr auf das Bob, das verschwinden soll.

MeinRastPort RastPortPtr auf den RastPort, von dem das Bob verschwinden soll.

MeinViewPort ViewPortPtr auf den ViewPort, von dem das Bob verschwinden soll.

Wenn ein Bob vom Bildschirm verschwunden ist (GELGONE im Feld flags des VSprite-Records ist gesetzt) und Sie es ab dem Zeitpunkt nicht mehr sehen möchten, so können Sie den Befehl REMBOB aus GfxMacros verwenden:

RemBob (MeinBob);

MeinBob	BobPtr auf das Bob, das ab sofort nicht mehr zu sehen sein soll.

REMBOB setzt einfach das BOBAWAY-Flag im flags-Feld des Bob-Records (was Sie natürlich auch selbst tun können) und GRAPHICS meint, es habe dieses Flag gesetzt und zeichnet das Bob beim nächsten DRAWGLIST-Befehl nicht mehr. Gerade für Spiele ist dieser Befehl äußerst praktisch.

4.7.1 Kollisionen mit Bobs

Bob-Kollisionen sind absolut identisch mit VSprite-Kollisionen und werden auch genauso gehandhabt. Sehen Sie bitte im entsprechenden Kapitel nach. Nun soll ein Beispielprogramm für Bobs kommen:

```
MODULE BobDemo;

FROM SYSTEM    IMPORT ADR,ADDRESS,BITSET,INLINE,LONGSET;
FROM Intuition IMPORT NewWindow,WindowPtr,OpenWindow,WindowFlagSet,customScreen,
                      CloseWindow,IDCMPFlags,IDCMPFlagSet,ViewAddress,NewScreen,
                      ScreenPtr,OpenScreen,CloseScreen,WindowFlags;
FROM Graphics  IMPORT ViewModes,ViewModeSet,Bob,ViewPortPtr,AddBob,GelsInfo,
                      RemIBob,SortGList,InitGels,MrgCop,LoadView,BobFlags,
                      BobFlagSet,CollTable,RastPortPtr,DrawGList,SetRGB4,
                      VSprite,VSpriteFlags,VSpriteFlagSet,SetCollision,
                      DoCollision,InitMasks,VSpritePtr;
FROM Exec      IMPORT AllocMem,MemReqs,MemReqSet,FreeMem,SetSignal;

TYPE CardPtr = POINTER TO CARDINAL;
     RandSet = SET OF [Ø..31];

VAR BobVx              : ARRAY[Ø..7] OF VSprite;
    BobVFirst,BobVLast: VSprite;
    BobBx              : ARRAY[Ø..7] OF Bob;
    MeinWindow         : WindowPtr;
    MeinScreen         : ScreenPtr;
    WindowDaten        : NewWindow;
    ScreenDaten        : NewScreen;
    MeinGelsInfo       : GelsInfo;
    VPort              : ViewPortPtr;
    RPort              : RastPortPtr;
    ChipBob            : ARRAY[Ø..7] OF CardPtr;
    i,j                : INTEGER;
    Buffer,Coll,Line   : ARRAY[Ø..7] OF ADDRESS;
```

```
PROCEDURE Rand (Mask: RandSet; VAR vs: VSprite);
  BEGIN
    IF (3 IN Mask) THEN RemIBob(vs.vsBob,RPort,VPort); END;
  END Rand;

PROCEDURE ChipCopy(source: CardPtr; VAR dest: CardPtr; size:
LONGCARD);
VAR ChipPtr: CardPtr;
    copied : LONGCARD;
  BEGIN
    ChipPtr:=AllocMem(size,MemReqSet{chip});
    dest:=ChipPtr;
    copied:=Ø;
    REPEAT
      ChipPtr^ :=source^ ;
      INC(ChipPtr,2); INC(source,2); INC(copied,2);
    UNTIL copied=size;
  END ChipCopy;

PROCEDURE BobDaten; (* $E- *)
  BEGIN
    INLINE(ØCF3CH,ØF3CFH);  (* 1.Bitplane *)
    INLINE(ØCF3CH,ØF3CFH);
    INLINE(Ø3CF3H,ØCF3CH);
    INLINE(Ø3CF3H,ØCF3CH);
    INLINE(ØF3CFH,Ø3CF3H);
    INLINE(ØF3CFH,Ø3CF3H);
    INLINE(ØCF3CH,ØF3CFH);
    INLINE(ØCF3CH,ØF3CFH);
    INLINE(Ø3CF3H,ØCF3CH);
    INLINE(Ø3CF3H,ØCF3CH);
    INLINE(ØF3CFH,Ø3CF3H);
    INLINE(ØF3CFH,Ø3CF3H);
    INLINE(ØCF3CH,ØF3CFH);
    INLINE(ØCF3CH,ØF3CFH);
    INLINE(Ø3CF3H,ØCF3CH);
    INLINE(Ø3CF3H,ØCF3CH);

    INLINE(Ø3CF3H,ØCF3CH); (* 2.Bitplane *)
    INLINE(Ø3CF3H,ØCF3CH);
    INLINE(ØF3FCH,Ø3CF3H);
    INLINE(ØF3CFH,Ø3CF3H);
    INLINE(ØCF3CH,ØF3CFH);
    INLINE(ØCF3CH,ØF3CFH);
    INLINE(Ø3CF3H,ØCF3CH);
```

```
    INLINE(Ø3CF3H,ØCF3CH);
    INLINE(ØF3CFH,Ø3CF3H);
    INLINE(ØF3CFH,Ø3CF3H);
    INLINE(ØCF3CH,ØF3CFH);
    INLINE(ØCF3CH,ØF3CFH);
    INLINE(Ø3CF3H,ØCF3CH);
    INLINE(Ø3CF3H,ØCF3CH);
    INLINE(ØF3CFH,Ø3CF3H);
    INLINE(ØF3CFH,Ø3CF3H);
  END BobDaten;

BEGIN
  FOR i:=Ø TO 7 DO
    ChipBob[i]:=NIL;
  END;

  FOR i:=Ø TO 7 DO
    ChipCopy(ADR(BobDaten),ChipBob[i],256);
  END;

  FOR i:=Ø TO 7 DO
    Buffer[i]:=AllocMem(192,MemReqSet{chip});
    (* 2 CARDINALs breit, 16 hoch, 3 (+planeOnOff) tief und alles *2 *)
    Coll[i]:=AllocMem(64,MemReqSet{chip});
    Line[i]:=AllocMem(4,MemReqSet{chip});
  END;

  FOR i:=Ø TO 7 DO
    WITH BobVx[i] DO
      width:=2; height:=16; depth:=2; imageData:=ChipBob[i];
      x:=11+i*((32Ø DIV 8)-8); y:=15+i*((256 DIV 8)-8);
      meMask:=BITSET{0}; hitMask:=BITSET{1}; planePick:=5;
      collMask:=Coll[i]; borderLine:=Line[i];
      planeOnOff:=Ø; flags:=VSpriteFlagSet{saveBack};
    END;
    WITH BobBx[i] DO
      flags:=BobFlagSet{}; saveBuffer:=Buffer[i];
      bobVSprite:=ADR(BobVx[i]);
    END;
    BobVx[i].vsBob:=ADR(BobBx[i]);
  END;

  WITH MeinGelsInfo DO
    sprRsrvd:=-1;
    nextLine:=AllocMem(16,MemReqSet{public,memClear});
    lastColor:=AllocMem(32,MemReqSet{public,memClear});
```

```
    collHandler:=AllocMem(SIZE(CollTable),MemReqSet{public,memClear});
    leftmost:=Ø; rightmost:=32Ø; topmost:=Ø; bottommost:=256;
  END;

  WITH ScreenDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=32Ø; height:=256; depth:=5;
    detailPen:=Ø; blockPen:=1; viewModes:=ViewModeSet{};
    type:=customScreen; font:=NIL; defaultTitle:=NIL; gadgets:=NIL;
    customBitMap:=NIL;
  END;

  WITH WindowDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=32Ø; height:=256; detailPen:=1;
    blockPen:=Ø; idcmpFlags:=IDCMPFlagSet{mouseButtons};
    flags:=WindowFlagSet{borderless}; firstGadget:=NIL;
    title:=NIL; screen:=NIL; bitMap:=NIL; checkMark:=NIL;
    type:=customScreen;
  END;

  MeinScreen:=OpenScreen(ScreenDaten);
  WindowDaten.screen:=MeinScreen;
  MeinWindow:=OpenWindow(WindowDaten);

  VPort:=ADR(MeinScreen^.viewPort);
  RPort:=ADR(MeinScreen^.rastPort);
  InitGels(ADR(BobVFirst),ADR(BobVLast),ADR(MeinGelsInfo));
  RPort^.gelsInfo:=ADR(MeinGelsInfo);

  SetCollision(Ø,ADR(Rand),ADR(MeinGelsInfo));

  FOR i:=Ø TO 7 DO
    InitMasks(ADR(BobVx[i]));
    AddBob(ADR(BobBx[i]),RPort);
  END;

  SetRGB4(VPort,7,Ø,Ø,Ø);
  SetRGB4(VPort,3,15,Ø,Ø);
  SetRGB4(VPort,6,15,15,15);

  WHILE NOT (MeinWindow^.userPort^.sigBit IN
    SetSignal(LONGSET{},LONGSET{})) DO
    FOR i:=Ø TO 7 DO
      INC(BobVx[i].x);
    END;
    SortGList(RPort);
    DoCollision(RPort);
    DrawGList(RPort,VPort);
```

```
    MrgCop(ViewAddress());
    LoadView(ViewAddress());
  END;

  CloseWindow(MeinWindow);
  CloseScreen(MeinScreen);
  FOR i:=Ø TO 7 DO
    FreeMem(ChipBob[i],256);
    FreeMem(Buffer[i],192);
  END;
END BobDemo.
```

4.8 Double-Buffering

Gerade bei relativ großen Bobs ist das menschliche Auge schneller als der Blitter und erkennt ein mehr oder minder starkes Flackern beim Verschieben von Bobs, was ja sicher nicht wünschenswert ist. Um das zu vermeiden, gibt es eine relativ einfache Technik: man erstellt für einen Screen zwei Bitmaps, zwischen denen bei jeder Bob-Bewegung umgeschaltet wird. Ist das Bob auf der ersten Bitmap zu sehen, wird es auf der zweiten an einer etwas verschobenen Position gezeichnet. So entsteht der Eindruck einer fließenden Bewegung. Name der Technik: Double-Buffering.

Da das System Double-Buffering nicht unterstützt, müssen wir das Erstellen und Verwalten unserer eigenen Bitmaps selbst übernehmen. Lediglich die Bobs verwaltet es in diesem Modus. Wir sollten bei der Erstellung einer Double-Buffer-Bitmap wie folgt vorgehen:

1. Initialisierung der Bitmap selbst
2. Initialisierung der Bitplanes
3. Initialisierung eines DBufPackets
4. DBufPacket in die Bob-Struktur eintragen

Zu 1: Die Initialisierung einer Bitmap übernimmt der Befehl INITBITMAP:

InitBitMap (MeineBitMap , Tiefe , Breite , Höhe);

MeineBitMap	BitMap-Record der zu initialisierenden Bitmap.
Tiefe	INTEGER-Wert, der die Anzahl der Bitplanes angibt.
Breite	INTEGER-Wert, der die Breite der Bitmap enthält.
Höhe	INTEGER-Wert, der die Höhe der Bitmap enthält.

Das ganze muß natürlich zweimal geschehen, da wir zwei Bitmaps initialisiert haben möchten.

Zu 2: Nun müssen wir für jede Bitmap eine bestimmte Anzahl an Planes bereitstellen. Den benötigten Speicher können wir mit ALLOCRASTER allozieren und den Zeiger darauf dem Feld planes übergeben. Die Bitplane sollte dann sofort gelöscht werden. Sind Sie mit der Initialisierung der Bitplanes fertig, sollten Sie einen Zeiger auf Ihre erste Bitmap dem Feld customBitMap des NewScreen-Records übergeben. Es ist zu beachten, daß im Feld type des NewScreen-Records unbedingt das Flag CUSTOMBITMAP gesetzt sein muß!

Zu 3 und 4: Damit auch die Bobs wissen, daß sie sich auf einem Double-Buffering-Display bewegen, muß jedem Bob-Record im Feld dBuffer ein Zeiger auf den DBufPacket-Record des Bobs übergeben werden. Dieser (er soll nicht näher besprochen werden, da er eigentlich keine interessanten Informationen birgt) ist wie folgt zu initialisieren:

MeinPacket = AllocMem (SIZE(DBufPacket),MemReqSet{memClear,chip});

MeinPacket DBufPacketPtr auf das zu initialisierende Packet.

Jedes Packet muß nun im Feld bufBuffer die Adresse eines speziellen Zwischenspeichers stehen haben, der genau die gleiche Größe hat wie der des Feldes saveBuffer. Übergeben Sie nun jedem Bob-Record einen Zeiger auf seine DBufPacket-Struktur, ist alles geregelt.

Nun können Sie zwischen den beiden Bitmaps hin- und herschalten, indem Sie die entsprechenden Zeiger des ViewPort- und RastPort-Records umbiegen:

```
MeinViewPort^.rasInfo^.bitmap := ADR(MeineBitmap1oder2);

MeinRastPort^.bitMap := ADR(MeineBitmap1oder2);

(MeinViewPort muß ein ViewPortPtr, MeinRastPort ein RastPortPtr sein.)
```

Der größte Teil des noch nicht behobenen Flackerns der Bobs kann nun durch den Befehl WAITTOF eliminiert werden. Nun dürfte es wohl kein Problem mehr sein, ein entsprechendes Programm zu schreiben:

```
MODULE DoubleBufferDemo;

FROM SYSTEM    IMPORT ADR,ADDRESS,BITSET,INLINE,LONGSET;
FROM Intuition IMPORT NewWindow,WindowPtr,OpenWindow,WindowFlagSet,customScreen,
                      CloseWindow,IDCMPFlags,IDCMPFlagSet,ViewAddress,NewScreen,
```

```
                        ScreenPtr,OpenScreen,CloseScreen,WindowFlags,ScreenFlags,
                        ScreenFlagSet;
FROM Graphics   IMPORT ViewModes,ViewModeSet,Bob,ViewPortPtr,AddBob,VSpritePtr,
                        RemIBob,SortGList,InitGels,MrgCop,LoadView,BobFlags,
                        BobFlagSet,CollTable,RastPortPtr,DrawGList,SetRGB4,
                        VSprite,VSpriteFlags,VSpriteFlagSet,SetCollision,GelsInfo,
                        DoCollision,InitMasks,DBufPacketPtr,InitBitMap,BitMap,
                        AllocRaster,DBufPacket,WaitTOF;
FROM Exec       IMPORT AllocMem,MemReqs,MemReqSet,FreeMem,SetSignal;

TYPE CardPtr = POINTER TO CARDINAL;
     RandSet = SET OF [Ø..31];

CONST
     ScrFlags = ScreenFlagSet{customBitMap};

VAR BobVx                     : ARRAY[Ø..7] OF VSprite;
    BobVFirst,BobVLast        : VSprite;
    BobBx                     : ARRAY[Ø..7] OF Bob;
    MeinWindow                : WindowPtr;
    MeinScreen                : ScreenPtr;
    WindowDaten               : NewWindow;
    ScreenDaten               : NewScreen;
    MeinGelsInfo              : GelsInfo;
    VPort                     : ViewPortPtr;
    RPort                     : RastPortPtr;
    ChipBob                   : ARRAY[Ø..7] OF CardPtr;
    i,j                       : INTEGER;
    Buffer,Coll,Line,Double: ARRAY[Ø..7] OF ADDRESS;
    Packet                    : ARRAY[Ø..7] OF DBufPacketPtr;
    MeineBitMap               : ARRAY[Ø..1] OF BitMap;

PROCEDURE Rand (Mask: RandSet; VAR vs: VSprite);
  BEGIN
    IF (3 IN Mask) THEN RemIBob(vs.vsBob,RPort,VPort); END;
  END Rand;

PROCEDURE ChipCopy(source: CardPtr; VAR dest: CardPtr; size:
LONGCARD);
VAR ChipPtr: CardPtr;
    copied : LONGCARD;
  BEGIN
    ChipPtr:=AllocMem(size,MemReqSet{chip});
    dest:=ChipPtr;
    copied:=Ø;
    REPEAT
```

```
      ChipPtr^:=source^;
      INC(ChipPtr,2); INC(source,2); INC(copied,2);
    UNTIL copied=size;
  END ChipCopy;

PROCEDURE BobDaten; (* $E- *)
  BEGIN
    INLINE(0CF3CH,0F3CFH);  (* 1.Bitplane *)
    INLINE(0CF3CH,0F3CFH);
    INLINE(03CF3H,0CF3CH);
    INLINE(03CF3H,0CF3CH);
    INLINE(0F3CFH,03CF3H);
    INLINE(0F3CFH,03CF3H);
    INLINE(0CF3CH,0F3CFH);
    INLINE(0CF3CH,0F3CFH);
    INLINE(03CF3H,0CF3CH);
    INLINE(03CF3H,0CF3CH);
    INLINE(0F3CFH,03CF3H);
    INLINE(0F3CFH,03CF3H);
    INLINE(0CF3CH,0F3CFH);
    INLINE(0CF3CH,0F3CFH);
    INLINE(03CF3H,0CF3CH);
    INLINE(03CF3H,0CF3CH);

    INLINE(03CF3H,0CF3CH); (* 2.Bitplane *)
    INLINE(03CF3H,0CF3CH);
    INLINE(0F3FCH,03CF3H);
    INLINE(0F3CFH,03CF3H);
    INLINE(0CF3CH,0F3CFH);
    INLINE(0CF3CH,0F3CFH);
    INLINE(03CF3H,0CF3CH);
    INLINE(03CF3H,0CF3CH);
    INLINE(0F3CFH,03CF3H);
    INLINE(0F3CFH,03CF3H);
    INLINE(0CF3CH,0F3CFH);
    INLINE(0CF3CH,0F3CFH);
    INLINE(03CF3H,0CF3CH);
    INLINE(03CF3H,0CF3CH);
    INLINE(0F3CFH,03CF3H);
    INLINE(0F3CFH,03CF3H);
  END BobDaten;

BEGIN
  FOR i:=0 TO 7 DO
    ChipBob[i]:=NIL;
  END;
```

```
FOR i:=Ø TO 7 DO
  ChipCopy(ADR(BobDaten),ChipBob[i],256);
END;

FOR i:=Ø TO 7 DO
  Buffer[i]:=AllocMem(192,MemReqSet{chip});
  Double[i]:=AllocMem(192,MemReqSet{chip});
  (* 2 CARDINALs breit, 16 hoch, 3 (+planeOnOff) tief und alles *2 *)

  Packet[i]:=AllocMem(SIZE(DBufPacket),MemReqSet{memClear,chip});
  Coll[i]:=AllocMem(64,MemReqSet{chip});
  Line[i]:=AllocMem(4,MemReqSet{chip});
END;

FOR i:=Ø TO 7 DO
  WITH BobVx[i] DO
    width:=2; height:=16; depth:=2; imageData:=ChipBob[i];
    x:=11+i*((32Ø DIV 8)-8); y:=15+i*((256 DIV 8)-8);
    meMask:=BITSET{0}; hitMask:=BITSET{1}; planePick:=5;
    collMask:=Coll[i]; borderLine:=Line[i];
    planeOnOff:=Ø; flags:=VSpriteFlagSet{saveBack};
  END;
  Packet[i]^.bufBuffer:=Double[i];
  WITH BobBx[i] DO
    flags:=BobFlagSet{}; saveBuffer:=Buffer[i];  bobVSprite:=ADR(BobVx[i]);
    dBuffer:=Packet[i];
  END;
  BobVx[i].vsBob:=ADR(BobBx[i]);
END;

WITH MeinGelsInfo DO
  sprRsrvd:=-1;
  nextLine:=AllocMem(16,MemReqSet{public,memClear});
  lastColor:=AllocMem(32,MemReqSet{public,memClear});
  collHandler:=AllocMem(SIZE(CollTable),MemReqSet{public,memClear});
  leftmost:=Ø; rightmost:=32Ø; topmost:=Ø; bottommost:=256;
END;

WITH ScreenDaten DO
  leftEdge:=Ø; topEdge:=Ø; width:=32Ø; height:=256; depth:=3; font:=NIL;
  detailPen:=Ø; blockPen:=1; viewModes:=ViewModeSet{}; gadgets:=NIL;
  type:=customScreen+ScrFlags; defaultTitle:=NIL;
  customBitMap:=NIL;
END;
```

```
WITH WindowDaten DO
  leftEdge:=Ø; topEdge:=Ø; width:=32Ø; height:=256; detailPen:=1;
  blockPen:=Ø; idcmpFlags:=IDCMPFlagSet{mouseButtons};
  flags:=WindowFlagSet{borderless}; firstGadget:=NIL;
  title:=NIL; screen:=NIL; bitMap:=NIL; checkMark:=NIL;
  type:=customScreen;
END;

InitBitMap(MeineBitMap[Ø],3,32Ø,256);
InitBitMap(MeineBitMap[1],3,32Ø,256);

FOR i:=Ø TO 1 DO
  FOR j:=Ø TO 3 DO
    MeineBitMap[i].planes[j]:=AllocRaster(32Ø,256);
  END;
END;
ScreenDaten.customBitMap:=ADR(MeineBitMap[Ø]);

MeinScreen:=OpenScreen(ScreenDaten);
WindowDaten.screen:=MeinScreen;
MeinWindow:=OpenWindow(WindowDaten);

VPort:=ADR(MeinScreen^.viewPort);
RPort:=ADR(MeinScreen^.rastPort);
InitGels(ADR(BobVFirst),ADR(BobVLast),ADR(MeinGelsInfo));
RPort^.gelsInfo:=ADR(MeinGelsInfo);

SetCollision(Ø,ADR(Rand),ADR(MeinGelsInfo));

FOR i:=Ø TO 7 DO
  InitMasks(ADR(BobVx[i]));
  AddBob(ADR(BobBx[i]),RPort);
END;

SetRGB4(VPort,7,Ø,Ø,Ø);
SetRGB4(VPort,3,15,Ø,Ø);
SetRGB4(VPort,6,15,15,15);

j:=1;

WHILE NOT (MeinWindow^.userPort^.sigBit IN SetSignal(LONGSET{},LONGSET{})) DO
  FOR i:=Ø TO 7 DO
    INC(BobVx[i].x);
  END;
  SortGList(RPort);
  DoCollision(RPort);
  DrawGList(RPort,VPort);
  WaitTOF;
```

```
    MrgCop(ViewAddress());
    LoadView(ViewAddress());
    VPort^.rasInfo^.bitMap:=ADR(MeineBitMap[j]);
    RPort^.bitMap:=ADR(MeineBitMap[j]);
    IF j=1 THEN j:=Ø; ELSE j:=1; END;
  END;

  CloseWindow(MeinWindow);
  CloseScreen(MeinScreen);
  FOR i:=Ø TO 7 DO
    FreeMem(ChipBob[i],256);
    FreeMem(Buffer[i],192);
    FreeMem(Double[i],192);
    FreeMem(Packet[i],SIZE(DBufPacket));
  END;
END DoubleBufferDemo.
```

4.9 Animationen

Na, auf dieses Kapitel wurde ja auch einige Male verwiesen, jetzt ist es soweit. Was steckt nun dahinter? Ein komplexes Animationssystem, das von GRAPHICS gesteuert wird. Gerade um »natürlichen« Objekten, wie Tieren z.B., mehr Realitätsnähe zu geben, müssen sich diese bewegen, und zwar nicht einfach nur in irgendwelche Richtungen, nein, die meisten Objekte verändern sich ja bei Bewegungen. Eine Katze gleitet schließlich nicht über den Boden, sie setzt ein Bein vor das andere. Mit dem Amiga läßt sich so ein Effekt erreichen, indem man das Objekt in verschiedenen Bewegungsphasen (Sequenzen) zeichnet und diese nacheinander ablaufen läßt. Speziell für Bobs gibt es nun ein System, das die von Ihnen gezeichneten Sequenzen automatisch wie in einem Film in festgelegter Reihenfolge ablaufen läßt, während Sie sich auf das Wesentliche konzentrieren können.

Zuerst sollten Sie sich die verschiedenen Bewegungssequenzen Ihres Objektes ausdenken und als Bobs definieren. Sind Sie soweit, müssen Sie Ihrer Bob-Struktur (genauer gesagt dem Feld bobComp) einen Zeiger auf einen AnimComp-Record zuweisen, der eine Animationskomponente (also einen Teil des Gesamtobjektes) definiert:

```
AnimComp=RECORD
  flags      : INTEGER     ; Flags für Animation
  timer      : INTEGER     ; Countdown
  timeSet    : INTEGER     ; Startwert für timer
  nextComp   : AnimCompPtr ; nächste Komponente
  prevComp   : AnimCompPtr ; vorherige Komponente
```

```
  nextSeq     : AnimCompPtr ; nächste Sequenz
  prevSeq     : AnimCompPtr ; vorherige Sequenz
  animCRoutine: ADDRESS     ; eigene Routine
  yTrans      : INTEGER     ; y-Koordinate der Position
  xTrans      : INTEGER     ; x-Koordinate der Position
  headOb      : AnimObPtr   ; Zeiger auf Objekt
  animBob     : BobPtr      ; Zeiger auf das betreffende Bob
END;
```

flags — Für dieses Feld ist eigentlich nur ein Flag sinnvoll: ringtrigger. Das RINGTRIGGER-Flag verbindet die einzelnen Animationssequenzen zu einem Ring, der kontinuierlich abläuft.

timer — Dieses Feld ist eigentlich ein Countdown der Dauer, für die das Objekt in dieser Phase zu sehen ist, denn der Wert des Feldes timeSet wird hierein kopiert und bis 0 heruntergezählt.

timeSet — In dieses Feld müssen Sie eintragen, wie lange das Objekt in dieser Phase zu sehen ist. Die Zeitangabe erfolgt hierbei in Animationsaufrufen, die wir aber gleich noch kennenlernen werden.

nextComp — Dieses Feld zeigt auf die nächste Animationskomponente.

prevComp — In diesem Feld steht ein Zeiger auf die vorherige Animationskomponente. Benutzen Sie RINGTRIGGER-Animationen, so ist dieses Feld wie auch nextComp auf NIL zu setzen.

nextSeq — Dieses Feld gibt die nächste Animationssequenz bei RINGTRIGGER-Animationen an. Als Animationssequenz wird hier eine Phase in der Animation verstanden.

prevSeq — Hier müssen Sie bei RINGTRIGGER-Animationen einen Zeiger auf die nächste Animationssequenz angeben.

animCRoutine — In diesem Feld können Sie die Adresse einer eigenen Routine, die bei jedem Animationsaufruf aufgerufen wird, wenn diese Sequenz gerade zu sehen ist, angeben. Als Parameter bekommt Ihre Routine den AnimComp-Record dieser Sequenz hinterhergeliefert. Möchten Sie keine solche Routine installieren, geben Sie hier am besten NIL an.

yTrans — Dieses Feld gibt die y-Koordinate der Komponenten-Position an. Da die unteren 6 Bit der INTEGER-Zahl für das System einen Nachkommateil bilden, müssen Sie Ihren Wert stets mit 64 multiplizieren, damit GRAPHICS ihn richtig versteht. Diese Position ist relativ zu der des zugehörigen Objektes.

xTrans Hier können Sie die x-Koordinate der Komponenten Position angeben. Die unteren 6 Bit dieser Zahl bilden für das System einen Nachkommateil, weshalb Sie Ihren Wert auch mit 64 multiplizieren müssen, damit er korrekt verstanden wird. Dieser Wert ist relativ zur Position des übergeordneten Objektes.

headOb In dieses Feld müssen Sie einen Zeiger auf die Struktur des Objektes der Sequenz angeben. Die Sequenz, die im AnimComp-Record definiert wird, ist ja schließlich nur ein Teil eines Gesamtobjektes, und genau auf dieses muß headOb zeigen.

animBob Hier müssen Sie einen Zeiger auf das Bob, das diese Sequenz darstellt, eintragen.

Nun zu den Objekten, die den Komponenten übergeordnet sind. Erstere werden mit der AnimOb-Struktur definiert:

```
AnimOb=RECORD
  nextOb      : AnimObPtr   ; Zeiger auf das nächste Objekt, wird vom System
                              verwaltet
  prevOb      : AnimObPtr   ; Zeiger auf das vorherige Objekt, wird vom System
                              verwaltet
  clock       : LONGINT     ; erlebte Animationsaufrufe
  anOldY      : INTEGER     ; y-Koordinate der alten Position
  anOldX      : INTEGER     ; x-Koordinate der alten Position
  anY         : INTEGER     ; y-Koordinate der Position im Festkommaformat
  anX         : INTEGER     ; x-Koordinate der Position im Festkommaformat
  yVel        : INTEGER     ; y-Geschwindigkeit
  xVel        : INTEGER     ; x-Geschwindigkeit
  yAccel      : INTEGER     ; y-Beschleunigung
  xAccel      : INTEGER     ; x-Beschleunigung
  ringYTrans  : INTEGER     ; y-Geschwindigkeit ohne Beschleunigung
  ringXTrans  : INTEGER     ; x-Geschwindigkeit ohne Beschleunigung
  animORoutine: ADDRESS     ; eigene Routine
  headComp    : AnimCompPtr ; erste Animationskomponente
END;
```

clock In dieses Feld schreibt GRAPHICS die Anzahl der Animationsaufrufe, die dieses Objekt schon erlebt hat.

anOldY-anX Diese Felder erklären sich wohl von selbst, doch ist zu sagen, daß die Werte darin im Festkommaformat stehen müssen. GRAPHICS wertet die unteren 6 Bit der Zahl als einen Nachkommateil aus, weshalb Sie Ihren Wert mit 64 multiplizieren müssen, bevor Sie ihn eintragen können. Doch einen Nachteil hat das Festkomma-

format: bedingt durch den Wertebereich einer INTEGER-Variablen können Sie nur 512 Punkte der Horizontalen ansprechen (512*64 = 32768, INTEGER-Grenze), doch mit einem Trick kann auch dieser Haken beseitigt werden: sollte Ihr Objekt die 512-Punkte-Grenze überschreiten, addieren Sie zu dem Feld xTrans der Anim-Comp-Records, die ja das eigentliche Aussehen des Objektes bestimmen, einen Wert bis 128, da die in xTrans gespeicherte Koordinate relativ zu der in anX angesehen wird. So können Sie mit dem Wert 512 in anX und 128 in xTrans recht einfach den 640sten Punkt der Horizontalen erreichen, denn 512 + 128 = 640!

yVel — Hier wird die Geschwindigkeit, mit der sich das Objekt in y-Richtung fortbewegen soll, angegeben. Bei jedem Animationsaufruf werden diese Werte zu denen der Felder anY und anX addiert. Da das System aber nicht gerade langsam ist (mehrere Animationsaufrufe pro Sekunde), werden Sie herzlich wenig von Ihrer mühevoll kreierten Animation haben. Deshalb wird dieser Wert als 64stel Animationsaufrufe ausgewertet, d.h. da Ihr Objekt nur alle 64 Animationsaufrufe bewegt wird. Das erklärt auch das Festkommaformat, in dem die Position angegeben werden muß.

xVel — In diesem Feld muß die Geschwindigkeit, mit der sich Ihr Objekt in x-Richtung fortbewegen soll, angegeben werden. Der Wert muß im Festkommaformat angegeben werden; warum siehe yVel.

yAccel — Hier muß die Beschleunigung des Objekts in y-Richtung stehen. Dieser Wert muß in 64stel Animationsaufrufen eingetragen werden, damit sich das Objekt nicht zu schnell fortbewegt. Dieser Wert wird zu yVel addiert, der wiederum zu anY hinzugerechnet wird. Durch Beschleunigung wirkt das Objekt nochmals natürlicher.

xAccel — In diesem Feld steht die Beschleunigung des Objekts in x-Richtung. Der Wert muß in 64stel Animationsaufrufen übergeben werden. Für nähere Informationen siehe yAccel.

ringYTrans — Wenn Sie auf Beschleunigungen verzichten möchten, sollten Sie in diesem Feld die Geschwindigkeit für die Bewegung in y-Richtung eintragen und die Felder yVel, xVel, yAccel und xAccel auf 0 setzen.

ringXTrans — Dieses Feld gibt die konstante Geschwindigkeit in x-Richtung an, wenn Sie auf Beschleunigungen verzichten möchten. Für weitere Informationen siehe ringYTrans.

animORoutine In dieses Feld können Sie bei Bedarf die Adresse einer eigenen Animationsprozedur, die bei jedem Animationsaufruf angesprungen wird, eintragen, ähnlich wie animCRoutine des AnimComp-Records. Ihrer Prozedur wird die Struktur des betreffenden Objektes übergeben. Der wesentliche Unterschied zur animCRoutine besteht darin, daß die animCRoutine nur bei jedem Animationsaufruf, bei dem die entsprechende Komponente zu sehen ist, angesprungen wird.

headComp Hier müssen Sie einen Zeiger auf die erste Komponente Ihres Objekts angeben.

Wenn Sie alle benötigten Records (GelsInfo, VSprite, Bob, AnimComp, AnimOb, diverse Zwischenspeicher und die Bilddaten) initialisiert haben, können Sie Ihre Objekte mit ADDANIMOB für die Bildschirmdarstellung vorbereiten (gezeichnet werden sie mit DRAWGLIST):

AddAnimOb (MeinObjekt , MeinKey , MeinRastPort);

MeinObjekt AnimObPtr auf den AnimOb-Record des Objekts, das vorbereitet werden soll.

MeinKey ADDRESSe des zuletzt vorbereiteten Objekts. Ist dies der erste ADDANIMOB-Aufruf, so muß hier NIL eingetragen werden.

MeinRastPort RastPortPtr auf den RastPort des Fensters (oder Screens), in dem das Objekt erscheinen soll.

Und nun zu den sagenumwobenen Animationsaufrufen: diese geschehen mit dem Befehl ANIMATE:

Animate (MeinKey , MeinRastPort);

MeinKey ADDRESSe des ersten AnimOb-Objekts.

MeinRastPort RastPortPtr auf den RastPort, dessen Animationsliste erneuert werden soll.

ANIMATE erneuert die Animationsliste eines RastPorts, d.h. dieser Befehl führt folgende Dinge durch:

- erneuert Position und Geschwindigkeit (bei Beschleunigung) jedes Objekts
- ruft die mit animORoutine bezeichnete Prozedur auf

- regelt bei jeder Komponente folgende Dinge:
  - schaltet zu einer neuen Sequenz um, wenn die vorherige vorbei ist
  - ruft die mit animCRoutine bezeichnete Prozedur auf
  - setzt die y- und x-Koordinaten der Bobs

Zudem gibt es noch ein Kommando aus GfxMacros, das eine Adresse auf NIL setzt (kann man auch »per Hand« erledigen):

InitAnimate (Adresse);

Adresse	ADDRESS-Wert, der auf NIL gesetzt werden soll.

Zum Schluß sei noch gesagt, daß Sie auch Kollisionen mit Animationsobjekten abfragen können, da diese ja aus Bobs bestehen. Initialisierung und Abfrage erfolgt in der Weise, wie Sie es im Kapitel über Bobs kennengelernt haben.

Zur Verdeutlichung dieser recht komplexen Materie soll nun ein Beispielprogramm kommen, bevor wir zu einem neuen Thema übergehen:

```
MODULE AnimDemo;

(* Dieses Programm generiert einen mampfenden Pac-Man, der sich quer über
   den Bildschirm bewegt. Die Animation besteht aus vier Sequenzen und
   somit vier Bobs:

   1. grinsender Pac-Man         2. Pac-Man mit halb offenem Mund
   3. Pac-Man mit offenem Mund   4. wie 2.

   Die vierte Sequenz war notwendig, da es sich um eine RINGTRIGGER-
   Animation handelt und der Ring geschlossen sein muß.
   Die einzelnen Sequenzen müssen, wie bekannt, durch prevSeq und nextSeq
   miteinander verbunden sein. Das prevSeq-Feld der ersten Sequenz muß
   hierbei auf die letzte zeigen, das nextSeq-Feld der letzten Sequenz wieder
   auf die erste. *)

FROM SYSTEM     IMPORT ADR,ADDRESS,BITSET,INLINE,LONGSET;
FROM Intuition  IMPORT NewWindow,WindowPtr,OpenWindow,WindowFlagSet,customScreen,
                       CloseWindow,IDCMPFlags,IDCMPFlagSet,NewScreen,
                       ScreenPtr,OpenScreen,CloseScreen,WindowFlags;
FROM Graphics   IMPORT ViewModes,ViewModeSet,Bob,ViewPortPtr,AddAnimOb,GelsInfo,
                       AnimComp,SortGList,InitGels,BobFlags,InitMasks,WaitTOF,
                       BobFlagSet,CollTable,RastPortPtr,DrawGList,SetRGB4,
                       VSprite,VSpriteFlags,VSpriteFlagSet,Animate,AnimOb;
FROM Exec       IMPORT AllocMem,MemReqs,MemReqSet,FreeMem,SetSignal;
```

```
TYPE CardPtr = POINTER TO CARDINAL;

VAR BobVx              : ARRAY[Ø..3] OF VSprite;
    BobVFirst,BobVLast: VSprite;
    BobBx              : ARRAY[Ø..3] OF Bob;
    MeinWindow         : WindowPtr;
    MeinScreen         : ScreenPtr;
    WindowDaten        : NewWindow;
    ScreenDaten        : NewScreen;
    MeinGelsInfo       : GelsInfo;
    VPort              : ViewPortPtr;
    RPort              : RastPortPtr;
    ChipBob            : ARRAY[Ø..3] OF CardPtr;
    i,j                : INTEGER;
    Buffer,Coll,Line   : ARRAY[Ø..3] OF ADDRESS;
    MeinAnim           : ARRAY[Ø..3] OF AnimComp;
    MeinObjekt,HeadOb  : AnimOb;

(* Diese Routine wird bei jedem Animationsaufruf angesprungen. Sie kontrolliert
    die Position des Pacmans und ändert, wenn nötig, die Fortbewegungsrichtung. *)

PROCEDURE ORoutine;
  BEGIN
    IF (MeinObjekt.anX < (-128*64)) OR (MeinObjekt.anX > ((512-48)*64)) THEN
      MeinObjekt.ringXTrans:=Ø-MeinObjekt.ringXTrans;
    END;
    IF (MeinObjekt.anY < Ø) OR (MeinObjekt.anY > (255*64)) THEN
      MeinObjekt.ringYTrans:=Ø-MeinObjekt.ringYTrans;
    END;
  END ORoutine;

PROCEDURE ChipCopy(source: CardPtr; VAR dest: CardPtr; size:
LONGCARD);
VAR ChipPtr: CardPtr;
    copied : LONGCARD;
  BEGIN
    ChipPtr:=AllocMem(size,MemReqSet{chip});
    dest:=ChipPtr;
    copied:=Ø;
    REPEAT
      ChipPtr^:=source^;
      INC(ChipPtr,2); INC(source,2); INC(copied,2);
    UNTIL copied=size;
  END ChipCopy;
```

```
(* Daten des ersten Bobs — ein Pac-Man mit breitem Grinsen *)

PROCEDURE BobDaten1; (* $E- *)
  BEGIN
    INLINE(ØØØ3FH,ØF8ØØH);
    INLINE(ØØ3CØH,ØØ78ØH);
    INLINE(ØØCØØH,ØØØ6ØH);
    INLINE(Ø1ØØØH,ØØE1ØH);
    INLINE(Ø6ØØØH,ØØEØCH);
    INLINE(Ø4ØØØH,ØØØØ4H);
    INLINE(Ø8ØØ3H,ØØØØ2H);
    INLINE(Ø8ØØ1H,ØCØØ2H);
    INLINE(Ø8ØØØH,Ø7EØ2H);
    INLINE(Ø4ØØØH,ØØ3FCH);
    INLINE(Ø6ØØØH,ØØØØCH);
    INLINE(Ø1ØØØH,ØØØ1ØH);
    INLINE(ØØCØØH,ØØØ6ØH);
    INLINE(ØØ3CØH,ØØF8ØH);
    INLINE(ØØØ3FH,ØF8ØØH);
    INLINE(ØØØØØH,ØØØØØH);
  END BobDaten1;

(* Daten des zweiten Bobs — ein Pac-Man mit halb offenem Mund *)

PROCEDURE BobDaten2; (* $E- *)
  BEGIN
    INLINE(ØØØ3FH,ØF8ØØH);
    INLINE(ØØ3CØH,ØØ78ØH);
    INLINE(ØØCØØH,ØØØ6ØH);
    INLINE(Ø1ØØØH,ØØE1ØH);
    INLINE(Ø6ØØØH,ØØEØCH);
    INLINE(Ø4ØØØH,ØØØØCH);
    INLINE(Ø8ØØØH,ØØØ3ØH);
    INLINE(Ø8ØØØH,ØØØCØH);
    INLINE(Ø8ØØØH,ØØØ3ØH);
    INLINE(Ø4ØØØH,ØØØØCH);
    INLINE(Ø6ØØØH,ØØØØCH);
    INLINE(Ø1ØØØH,ØØØ1ØH);
    INLINE(ØØCØØH,ØØØ6ØH);
    INLINE(ØØ3CØH,ØØF8ØH);
    INLINE(ØØØ3FH,ØF8ØØH);
    INLINE(ØØØØØH,ØØØØØH);
  END BobDaten2;

(* Daten des dritten Bobs — ein Pac-Man mit offenem Mund *)
```

```
PROCEDURE BobDaten3; (* $E- *)
  BEGIN
    INLINE(0003FH,0F800H);
    INLINE(003C0H,00780H);
    INLINE(00C00H,00060H);
    INLINE(01000H,00E20H);
    INLINE(06000H,001C0H);
    INLINE(04000H,01E00H);
    INLINE(08001H,0E000H);
    INLINE(0800EH,00000H);
    INLINE(08001H,0E000H);
    INLINE(04000H,01E00H);
    INLINE(06000H,001C0H);
    INLINE(01000H,00020H);
    INLINE(00C00H,00060H);
    INLINE(003C0H,00F80H);
    INLINE(0003FH,0F800H);
    INLINE(00000H,00000H);
  END BobDaten3;

BEGIN
  FOR i:=0 TO 3 DO
    ChipBob[i]:=NIL;
  END;

  ChipCopy(ADR(BobDaten1),ChipBob[0],64); (* Bob für Sequenz 1 *)
  ChipCopy(ADR(BobDaten2),ChipBob[1],64); (* Bob für Sequenz 2 *)
  ChipCopy(ADR(BobDaten3),ChipBob[2],64); (* Bob für Sequenz 3 *)
  ChipCopy(ADR(BobDaten2),ChipBob[3],64); (* Bob für Sequenz 2 *)

  FOR i:=0 TO 3 DO
    Buffer[i]:=AllocMem(64,MemReqSet{chip});
    (* 2 CARDINALs breit, 16 hoch, 1 tief und alles *2 *)
    Coll[i]:=AllocMem(64,MemReqSet{chip});
    Line[i]:=AllocMem(6,MemReqSet{chip});
  END;

  FOR i:=0 TO 3 DO
    WITH BobVx[i] DO
      width:=2; height:=16; depth:=1; imageData:=ChipBob[i];
      collMask:=Coll[i]; borderLine:=Line[i]; planePick:=1;
      planeOnOff:=0; flags:=VSpriteFlagSet{saveBack};
      y:=32*i; x:=0;
    END;
    WITH BobBx[i] DO
      flags:=BobFlagSet{bobIsComp}; bobVSprite:=ADR(BobVx[i]);
```

```
      imageShadow:=Coll[i]; bobComp:=ADR(MeinAnim[i]); saveBuffer:=Buffer[i];
    END;
    BobVx[i].vsBob:=ADR(BobBx[i]);
  END;

(* Ab hier werden die vier Komponenten der Animation initialisiert. Jede
   Komponente hat ihr eigenes Bob:

   Komponente  1  2  3  4
   Bob         1  2  3  2

   Jede Komponente (außer 1) ist außerdem 6 Animationsaufrufe lang
   sichtbar (timeSet=6) und bewegt sich mit einer Geschwindigkeit von 128
   Animationsaufrufen.*)

  WITH MeinAnim[Ø] DO
    flags:=1H; timeSet:=9; nextSeq:=ADR(MeinAnim[1]);
    prevSeq:=ADR(MeinAnim[3]);
    yTrans:=Ø; xTrans:=128*64; headOb:=ADR(MeinObjekt);
    animBob:=ADR(BobBx[Ø]);
  END;

  WITH MeinAnim[1] DO
    flags:=1H; timeSet:=6; nextSeq:=ADR(MeinAnim[2]);
    prevSeq:=ADR(MeinAnim[2]);
    yTrans:=Ø; xTrans:=128*64; headOb:=ADR(MeinObjekt);
    animBob:=ADR(BobBx[1]);
  END;

  WITH MeinAnim[2] DO
    flags:=1H; timeSet:=6; nextSeq:=ADR(MeinAnim[3]);
    prevSeq:=ADR(MeinAnim[3]);
    yTrans:=Ø; xTrans:=128*64; headOb:=ADR(MeinObjekt);
    animBob:=ADR(BobBx[2]);
  END;

  WITH MeinAnim[3] DO
    flags:=1H; timeSet:=6; nextSeq:=ADR(MeinAnim[Ø]);
    prevSeq:=ADR(MeinAnim[2]);
    yTrans:=Ø; xTrans:=128*64; headOb:=ADR(MeinObjekt);
    animBob:=ADR(BobBx[1]);
  END;

(* Initialisierung des Objektes, das sich aus den einzelnen Komponenten
   zusammensetzt. *)
```

```
WITH MeinObjekt DO
  anY:=Ø; anX:=Ø; ringYTrans:=64; ringXTrans:=128;
  headComp:=ADR(MeinAnim[Ø]); animORoutine:=ADR(ORoutine);
END;

WITH MeinGelsInfo DO
  sprRsrvd:=-1;
  nextLine:=AllocMem(16,MemReqSet{public,memClear});
  lastColor:=AllocMem(32,MemReqSet{public,memClear});
  collHandler:=AllocMem(SIZE(CollTable),MemReqSet{public,memClear});
  leftmost:=Ø; rightmost:=64Ø; topmost:=Ø; bottommost:=256;
END;

WITH ScreenDaten DO
  leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256; depth:=3;
  detailPen:=Ø; blockPen:=1; viewModes:=ViewModeSet{hires};
  type:=customScreen; font:=NIL; defaultTitle:=NIL; gadgets:=NIL;
  customBitMap:=NIL;
END;

WITH WindowDaten DO
  leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256; detailPen:=1;
  blockPen:=Ø; idcmpFlags:=IDCMPFlagSet{mouseButtons};
  flags:=WindowFlagSet{borderless}; firstGadget:=NIL;
  title:=NIL; screen:=NIL; bitMap:=NIL; checkMark:=NIL;
  type:=customScreen;
END;

MeinScreen:=OpenScreen(ScreenDaten);
WindowDaten.screen:=MeinScreen;
MeinWindow:=OpenWindow(WindowDaten);

VPort:=ADR(MeinScreen^.viewPort);
RPort:=ADR(MeinScreen^.rastPort);
InitGels(ADR(BobVFirst),ADR(BobVLast),ADR(MeinGelsInfo));
RPort^.gelsInfo:=ADR(MeinGelsInfo);

FOR i:=Ø TO 2 DO
  InitMasks(ADR(BobVx[i]));
END;

SetRGB4(VPort,Ø,Ø,Ø,Ø);
SetRGB4(VPort,1,15,15,15);

AddAnimOb(ADR(MeinObjekt),ADR(HeadOb),RPort); (* Objekt vorbereiten *)

WHILE NOT (MeinWindow^.userPort^.sigBit IN SetSignal(LONGSET{},LONGSET{})) DO
  Animate(ADR(HeadOb),RPort); (* Animationsaufruf !! *)
```

```
    SortGList(RPort);
    WaitTOF;
    DrawGList(RPort,VPort);
  END;

  CloseWindow(MeinWindow);
  CloseScreen(MeinScreen);
  FOR i:=0 TO 3 DO
    FreeMem(ChipBob[i],64);
    FreeMem(Buffer[i],64);
    FreeMem(Line[i],6);
    FreeMem(Coll[i],64);
  END;
END AnimDemo.
```

4.10 Dual-Playfield

Neben dem schon besprochenen Double-Buffering gibt es noch einen weiteren Spezialmodus, der es aber in sich hat: Dual-Playfield. Diese Darstellungsart macht es möglich, zwei völlig verschiedene Bitmaps (Playfields) übereinanderzulegen, wobei man durch »Löcher« im ersten Playfield (Teile, die in der Hintergrundfarbe gezeichnet sind) das zweite beobachten kann. Für diesen Modus sind folgende Vorbereitungen nötig:

1. Initialisieren eines eigenen Screens (Playfield 1)
2. Initialisieren einer zweiten Bitmap
3. Initialisieren eines zweiten RastPorts
4. Initialisieren eines zweiten RasInfos
5. Erneuern des Displays

Zu 1: Der Screen wird ganz normal definiert (das ViewMode-Flag DUALPF wird nicht gesetzt!) und geöffnet.

Zu 2: Eine zweite Bitmap wird mit dem Befehl INITBITMAP initialisiert und mit ALLOCRASTER Speicherplatz für die Planes (maximal 3!) bereitgestellt.

Zu 3: Ein zweiter RastPort kann mit dem Befehl INITRASTPORT initialisiert werden:

InitRastPort (MeinRastPort);

MeinRastPortRast Port-Record, der initialisiert werden soll.

Dann muß die Adresse der zweiten Bitmap in das Feld bitMap des zweiten RastPorts geschrieben werden, um diesem klarzumachen, daß er fortan für die zweite Bitmap zu sorgen hat.

Zu 4: Ihren zweiten RasInfo können Sie ganz normal mit einer WITH-Verschachtelung initialisieren. Wichtig ist, daß das Feld bitMap auf Ihre zweite Bitmap zeigen muß.

Nun müssen Sie den RasInfo an den Ihres Intuition-Screens anhängen, indem Sie einen Zeiger auf ihn in das Feld rasInfo des ViewPorts schreiben.

Zu 5: Jetzt, wo Sie Ihr zweites Playfield wunderbar an Intuition vorbei aufgebaut haben, müssen Sie das Flag DUALPF (und PFBA bei Bedarf) im Feld viewModes Ihres Screens setzen. Danach sollten Sie dem System klarmachen, daß Sie etwas an Ihrem Screen verändert haben, was am besten mit dem Befehl MAKESCREEN aus Intuition, der den Screen neu aufbaut, zu tun ist:

MakeScreen (MeinScreen);

MeinScreen	ScreenPtr auf den neu aufzubauenden Screen.

Haben Sie das getan, sollten Sie zur Erneuerung des Displays den Befehl REMAKEDISPLAY aus Intuition benutzen:

RemakeDisplay;

Herzlichen Glückwunsch, Ihr »second playfield« ist bereit. Nun können Sie es mit den Zeichenbefehlen richtig schön vollmalen. Geben Sie einfach Ihren Alternativ-RastPort als Zeichen-RastPort an.

Was Sie noch beachten sollten, ist die Verteilung der Farbregister im Dual-Playfield-Modus, da für beide Playfields nur ein ViewPort und somit eine ColorMap existiert: Die Register 0–7 gelten, unabhängig von dessen Bitplane-Tiefe, nur für Playfield 1. Für Playfield 2 hingegen sind ausschließlich die Farben 8–15 zu gebrauchen.

Damit auch dieser Stoff »sitzt«, sehen Sie sich das Ganze mal in Programmform an:

```
MODULE DPFDemo;

FROM SYSTEM     IMPORT ADR,ADDRESS,INLINE;
FROM Intuition  IMPORT ViewAddress,NewScreen,ScreenPtr,OpenScreen,CloseScreen,
                       customScreen,RethinkDisplay,MakeScreen,RemakeDisplay;
FROM Graphics   IMPORT AllocRaster,SetRast,SetAPen,Move,RectFill,Draw,SetRGB4,
                       FreeRaster,BitMap,RastPort,ViewModes,ViewModeSet,
                       RasInfo,InitRastPort,InitBitMap;
FROM Dos        IMPORT Delay;

VAR BMap        : BitMap;     (* zweite Bitmap *)
    Info        : RasInfo;    (* zweiter RasInfo *)
    RP          : RastPort;   (* zweiter RastPort *)
```

```
    ScreenDaten: NewScreen;
    MeinScreen : ScreenPtr;
    i,j        : INTEGER;

BEGIN
  WITH ScreenDaten DO
    leftEdge:=0; topEdge:=0; width:=640; height:=256; depth:=2;
    detailPen:=0; blockPen:=1; viewModes:=ViewModeSet{hires};
    type:=customScreen; customBitMap:=NIL; gadgets:=NIL; font:=NIL;
    defaultTitle:=ADR("Blau=Playfield 1; Grün=Playfield 2");
  END;

  MeinScreen:=OpenScreen(ScreenDaten);

  InitBitMap(BMap,1,640,256);            (* zweite Bitmap initialisieren *)
  BMap.planes[0]:=AllocRaster(640,256); (* 1 Bitplane allozieren *)

  InitRastPort(RP);                      (* zweiten RastPort initialisieren *)
  RP.bitMap:=ADR(BMap);                  (* zweiten RastPort anpassen ... *)
  SetRast(ADR(RP),0);                    (* ... und löschen! *)

  WITH Info DO    (* zweiten RasInfo initialisieren ... *)
    bitMap:=ADR(BMap); rxOffset:=0; ryOffset:=0; next:=NIL;
  END;

  MeinScreen^.viewPort.rasInfo^.next:=ADR(Info); (* ... und einbinden *)

  MeinScreen^.viewPort.modes:=ViewModeSet{hires,dualpf};
  MakeScreen(MeinScreen);
  RemakeDisplay;

  SetRGB4(ADR(MeinScreen^.viewPort),9,0,15,0); (* Farbe für 2. Playfield *)
        (* setzen *)
  FOR i:=0 TO 255 BY 40 DO
    FOR j:=0 TO 639 BY 40 DO
      Move(ADR(RP),0,i);
      Draw(ADR(RP),j,0);
    END;
  END;
  Delay(300);

  MeinScreen^.viewPort.rasInfo^.next:=NIL;         (* nur noch 1 Playfield *)
  MeinScreen^.viewPort.modes:=ViewModeSet{hires}; (* Normalmodus *)

  MakeScreen(MeinScreen);
  RethinkDisplay;       (* setzen! *)

  CloseScreen(MeinScreen);       (* Screen schließen *)
  FreeRaster(BMap.planes[0],640,256);      (* Plane freigeben *)
END DPFDemo.
```

4.10.1 Playfield-Scrolling

Und nun kommt das Beste, was man mit Playfields anstellen kann: Smooth-Scrolling. Smooth-Scrolling wird erreicht, indem Sie ein übergroßes Playfield (640*512 Punkte ohne Interlace, z.B.) definieren, es unter ein anderes legen und dann die rxOffset und ryOffset-Werte des entsprechenden RasInfos verändern. Playfield 1 sollte dabei ein »Loch« haben (also eine oder mehrere Stellen, die in transparenter Farbe gezeichnet worden sind), durch das Sie Playfield 2 scrollen sehen können. Sollte in Playfield 1 noch irgendeine Wahnsinns-Grafik (irgendein tolles Bild, Dortmunder Sonnenuntergang meinetwegen) zu sehen sein, kommt der Effekt auf ...

Um das allerdings zu realisieren (Entschuldigung, daß ich Sie aus Ihren Gedanken gerissen haben), sollten Sie beachten, daß Sie nach jeder Änderung der RasInfo-Felder rxOffset und ryOffset des Display mit REMAKEDISPLAY erneuern. Auch hierzu ein Beispielprogramm (nach einem Public-Domain-C-Programm von Gregg Williams:

```
MODULE DPFDemo;

(* Dieser Code liegt einem Public-Domain-Programm von GREGG WILLIAMS
   in C zugrunde. Es mußte für Intuition-Verhältnisse angepaßt werden,
   was sich als gar nicht so leicht erwies (2 Tage!) ...
   Playfield 1 ist ein ganz normaler Intuition-Screen, Playfield 2
   dagegen eine übergroße Bitmap (64Ø*512 Punkte ohne Interlace), die
   lustig durch die Gegend scrollt. *)

FROM SYSTEM     IMPORT ADR,ADDRESS,INLINE;
FROM Intuition  IMPORT ViewAddress,NewScreen,ScreenPtr,OpenScreen,CloseScreen,
                       customScreen,RethinkDisplay,MakeScreen,RemakeDisplay;
FROM Graphics   IMPORT AllocRaster,WaitTOF,SetRast,SetAPen,Move,RectFill,Draw,
                       FreeRaster,BitMap,RastPort,ViewModes,ViewModeSet,
                       LoadRGB4,RasInfo,InitRastPort,InitBitMap;

VAR BMap        : BitMap;                   (* 2. Bitmap *)
    Info        : ARRAY[Ø..1] OF RasInfo;   (* 1. + 2. RasInfo *)
    RP          : ARRAY[Ø..1] OF RastPort;  (* 1. + 2. RastPort *)
    nmax,n,i,x,y: INTEGER;
    ScreenDaten : NewScreen;
    MeinScreen  : ScreenPtr;

PROCEDURE Farben; (* $E- *)
  BEGIN
    INLINE(ØØØØH,Ø55FH,ØØØØH,ØC57H,ØØØØH,ØØØØH,ØØØØH,ØØØØH);
    INLINE(ØØØØH,ØFBØH,ØCCCH,ØF15H,ØA5ØH,ØØC5H,Ø777H,Ø77FH);
    INLINE(ØØØØH,ØØØØH,ØØØØH,ØØØØH,ØØØØH,ØØØØH,ØØØØH,ØØØØH);
    INLINE(ØØØØH,ØØØØH,ØØØØH,ØØØØH,ØØØØH,ØØØØH,ØØØØH,ØØØØH);
  END Farben;
```

```
PROCEDURE MakeBox(x1,y1,col1,col2: INTEGER);
  BEGIN
    SetAPen(ADR(RP[1]),0); (* Schatten zeichnen *)
    RectFill(ADR(RP[1]),x1+4,y1+4,x1+40,y1+40);
    SetAPen(ADR(RP[1]),0); (* Umrandung zeichnen *)
    RectFill(ADR(RP[1]),x1,y1,x1+36,y1+36);
    SetAPen(ADR(RP[1]),col1); (* großen Kasten zeichnen *)
    RectFill(ADR(RP[1]),x1+2,y1+2,x1+34,y1+34);
    SetAPen(ADR(RP[1]),col2); (* kleinen Kasten zeichnen *)
    RectFill(ADR(RP[1]),x1+12,y1+12,x1+24,y1+24);
  END MakeBox;

PROCEDURE ZeichneFields;
VAR x1,y1,temp,col1,col2: INTEGER;
  BEGIN
    SetRast(ADR(RP[0]),1);     (* Fenster auf 1. Playfield malen *)
    SetAPen(ADR(RP[0]),2);
    RectFill(ADR(RP[0]),95,45,225,183);
    SetAPen(ADR(RP[0]),0);
    RectFill(ADR(RP[0]),100,50,220,178);

    SetRast(ADR(RP[1]),15);
    SetAPen(ADR(RP[1]),14);

    temp:=1;
    FOR x1:=0 TO 640 BY 20 DO (* Übergroßes Playfield 1 mit Gitter füllen *)
      Move(ADR(RP[1]),x1,0);
      Draw(ADR(RP[1]),x1,512);
    END;
    FOR y1:=0 TO 512 BY 20 DO
      Move(ADR(RP[1]),0,y1);
      Draw(ADR(RP[1]),640,y1);
    END;

    FOR x1:=25 TO 590 BY 50 DO (* Übergroßes Playfield 1 mit Kästen füllen *)
      FOR y1:=0 TO 462 BY 50 DO
        col1:=temp+8;
        INC(temp);
        IF temp>7 THEN temp:=1; END;
        col2:=temp+8;
        INC(temp);
        IF temp>7 THEN temp:=1; END;
        MakeBox(x1,y1,col1,col2);
      END;
    END;
  END ZeichneFields;
```

```
BEGIN
  nmax:=1Ø; (* Wieviele Male soll Playfield 2 "anecken" ? *)

  WITH ScreenDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=322; height:=256; depth:=3;
    detailPen:=Ø; blockPen:=1; viewModes:=ViewModeSet{};
    type:=customScreen; defaultTitle:=NIL; gadgets:=NIL; font:=NIL;
    customBitMap:=NIL;
  END;

  MeinScreen:=OpenScreen(ScreenDaten);

  InitBitMap(BMap,3,64Ø,512); (* Bitmap 2 initialisieren *)

  FOR i:=Ø TO 2 DO
    BMap.planes[i]:=AllocRaster(64Ø,512); (* Bitplanes reservieren *)
  END;

  InitRastPort(RP[1]);          (* RastPort 2 erstellen *)
  RP[1].bitMap:=ADR(BMap);         (* Bitmap 2 einbinden *)

  WITH Info[1] DO          (* RasInfo 2 ausfüllen *)
    bitMap:=ADR(BMap); rxOffset:=Ø; ryOffset:=Ø; next:=NIL;
  END;

  RP[Ø]:=MeinScreen^.rastPort;          (* RastPort u. RasInfo deklarieren *)
  Info[Ø]:=MeinScreen^.viewPort.rasInfo^;
  MeinScreen^.viewPort.rasInfo^.next:=ADR(Info[1]); (* RasInfo 2 anpassen *)

  LoadRGB4(ADR(MeinScreen^.viewPort),ADR(Farben),32); (* Farben laden *)

  MeinScreen^.viewPort.modes:=ViewModeSet{dualpf}; (* DUALPF setzen *)
  MakeScreen(MeinScreen); (* erneuern *)
  RethinkDisplay;

  ZeichneFields;    (* Playfields füllen *)

  x:=1; y:=1;

  REPEAT
    Info[1].rxOffset:=Info[1].rxOffset+x; (* Koordinaten f. PF2 erhöhen *)
    Info[1].ryOffset:=Info[1].ryOffset+y;

    IF (Info[1].rxOffset<=Ø) OR (Info[1].rxOffset>=319) THEN
      x:=-x; INC(n);
    END;
    IF (Info[1].ryOffset<=Ø) OR (Info[1].ryOffset>=255) THEN
      y:=-y; INC(n);
    END;
    RemakeDisplay;          (* Display erneuern *)
  UNTIL n=nmax;
```

```
  MeinScreen^.viewPort.rasInfo^.next:=NIL;    (* Intuition-Screen setzen *)
  MeinScreen^.viewPort.modes:=ViewModeSet{}; (* DUALPF löschen *)

  MakeScreen(MeinScreen);          (* OK! *)
  RemakeDisplay;
  RethinkDisplay;

  CloseScreen(MeinScreen);
  FOR i:=Ø TO 2 DO          (* 2. Bitmap freigeben *)
    FreeRaster(BMap.planes[i],64Ø,512);
  END;
END DPFDemo.
```

4.11 Der Standard – IFF

IFF (Interchange File Format, Datenaustauschformat) ist ein von den Firmen Electronic Arts (die uns Games wie Bard's Tale I und II oder Anwendungsprogramme wie DPaint II, Deluxe Music und vieles andere mehr brachte) und Commodore ins Leben gerufenes Datenformat, das es ermöglicht, Daten zwischen verschiedenen Programmen auszutauschen, vorausgesetzt, diese Programme halten sich an den IFF-Standard, was aber fast immer der Fall ist. Inzwischen kamen verschiedene private IFF-Standards hinzu, die beispielsweise alternativ zu FTXT von den einzelnen Textverarbeitungen verschiedener Hersteller verwendet werden. Jedoch wird hier nicht auf diese (teils etwas exotischen) Individuallösungen eingegangen. Es gibt folgende verbreitete IFF-Untergruppen:

- SMUS (Musikdaten für Musikprogramme wie Sonix oder DeluxeMusic)
- FTXT (Formatierter Text)
- ANIM (Animationsdateien für z.B. VideoScape 3D)
- 8SVX (Digitalisierte Instrumente für Musikprogramme)
- ILBM (Grafikbilder von DPaint, GraphicCraft usw.)
- und noch viele andere weniger bekannte und benutzte (PDEF, Page Definition von DeluxePrint, z.B.)

Uns interessiert nun das ILBM-Format, das es möglich macht, z.B. Bilder zwischen verschiedenen Malprogrammen auszutauschen (Informationen über die anderen können Sie im ROM-KERNEL-MANUAL, Exec-Reference-Manual finden).

Den ILBM gibt es in zwei Varianten: 1. Das »reine« ILBM-Format, und 2. das »gepackte« ILBM-Format, das mit dem ByteRun-1-Algorithmus kodiert wurde und mehr oder minder viel Diskettenspeicherplatz spart. DPaint nutzt dieses Format.

Wie ist so ein ILBM-Bild aufgebaut? Die folgende Abbildung zeigt es (die Reihenfolge muß übrigens nicht so sein, fest steht aber, daß der BODY-Block am Ende des Files steht):

```
FORMxxxx — FORM-Chunk und Länge der Datei (Longword, 2 Words)

    xxxx — Kennung, um was für eine IFF-Datei es sich handelt
           (4 Byte), bei Bildern ILBM

BMHDxxxx — BMHD-Chunk und Länge des Chunks (Longword; eigentlich
           immer 2Ø)

         — Höhe der Grafik (Word)

         — Breite der Grafik (Word)

         — x-Position der Grafik (Word)

         — y-Position der Grafik (Word)

         — Anzahl der Bitplanes (Byte)

         — Masking (Byte): Ø = kein Masking,
                           1 = Masking,
                           2 = Transparent,
                           3 = Lassoing

         — Kompressionsart (Byte): Ø = keine Kompression,
                                   1 = ByteRun-1-Kompression

         — unbenutzt (Byte)

         — transparente Farbe (Word)
         — x-Aspect (Byte)

         — y-Aspect (Byte)

         — Seitenbreite (Word)

         — Seitenhöhe (Word)

CMAPxxxx — CMAP-Chunk und dessen Länge (Longword), die die Anzahl
           der verwendeten Farben angibt

         — Farbe Ø: Rotkomponente (Byte)
                    Grünkomponente (Byte)
                    Blaukomponente (Byte)

         — Farbe 1: Rotkomponente (Byte)
                    Grünkomponente (Byte)
                    Blaukomponente (Byte)

           ... (je nach Bitplane-Tiefe)
```

CRNGxxxx — CRNG-Chunk, 12 Byte Graphicraft-spezifischer und absolut unwichtiger Daten

DPPVxxxx — DPPV-Chunk, 1Ø8 Byte Graphicraft-spezifischer und absolut unwichtiger Daten

CAMGxxxx — CAMG-Chunk, dessen Länge (Londword)

— ViewPort-Modi der Grafik in Zahlen (Longword; vergessen Sie diesen Chunk im Moment)

CCRTxxxx — CCRT-Chunk und dessen Länge (Longword), der Spezialinformationen für die Farbanimation, das sog. "Color-Cycling" angibt

— Farbscrollrichtung (Word): Ø = rückwärts
1 = vorwärts

— Farbregister, ab dem gescrollt werden soll (Byte)

— Farbregister, bis zu dem gescrollt werden soll (Byte)

— benötigte Zeit, um von einem zum anderen Farbregister zu wechseln (Sekunden; Longword)

— benötigte Zeit, um von einem zum anderen Farbregister zu wechseln (Mikrosekunden; Longword)

BODYxxxx — BODY-Chunk und dessen Länge (Longword)

— hier stehen die Datenbytes der einzelnen Planes (Byte-Format) reihenweise geordnet, d.h. die erste Bitplane der ersten Zeile, die zweite Bitplane der ersten Zeile, ..., die erste Bitplane der zweiten Zeile, usw.

Als Chunks werden übrigens IFF-Datenblöcke bezeichnet.

Ich habe die Länge der einzelnen Werte in Klammern geschrieben, da sie für das Einlesen einer ILBM-Datei sehr wichtig sind, denn Sie müssen beim DOS-Befehl READ, mit dem das Einlesen erfolgt, ja immer die Anzahl der einzulesenden Bytes angeben. Hier nochmal die einzelnen Längen und Modula-2-Äquivalente:

Name	Länge			Modula-2-Äquivalent
Longword	= 2 Words	= 2*2 Byte	= 4 Byte	LONGCARD oder LONGINT
Word	= 1 Word	= 2*1 Byte	= 2 Byte	CARDINAL oder INTEGER
Byte	= Ø.5 Words	= 1*1 Byte	= 1 Byte	BYTE oder [Ø..255]

Nun stellt sich einem natürlich die Frage, wie man ein doch wohl so kompliziertes Bild einliest und dabei alles richtig auswertet. Eigentlich ist das gar nicht schwer und das

IFF-Format auch nicht so katastrophal kompliziert, man muß lediglich die Längen der Daten berücksichtigen.

Als Grafikunterlage soll uns ein ganz normaler Intuition-Screen dienen, den wir aber noch nicht fest bestimmen können, da wir ja nicht wissen, in welcher Auflösung das Bild gezeichnet wurde und wieviele Farben es hat. Also werden nur die Felder detailPen, blockPen und type konkret definiert, da diese Daten nicht bildspezifisch sind.

Für das eigentliche Einlesen der Daten definieren wir am besten eine WHILE-Schleife und ein BOOLEAN-Flag, das auf TRUE gesetzt wird, wenn die letzten Daten (und das ist auf jeden Fall der BODY Chunk) eingelesen und ausgewertet wurden. Die WHILE-Schleife dauert nun solange an, bis das BOOLEAN-Flag nicht mehr FALSE, sondern TRUE ist. Innerhalb dieser Schleife werden nun immer vier Bytes, die den Chunknamen angeben, eingelesen. Dieser Name kann nun ganz einfach mit den vorkommenden Codes verglichen werden und entsprechenden Reaktionen vom Programm hervorrufen. Es ist hierbei sinnvoll, einen Abfallspeicher einzurichten, in den die nicht benötigten Daten eingelesen werden können. Dessen Größe sollte gleich der des größten einzulesenden Datenblocks sein, und das ist wohl der DPPV-Chunk mit 108 Byte. In diesem Speicher sollten Sie vor der WHILE-Anweisung noch die 12 Byte, die den Dateikopf angeben (FORM, Gesamtlänge und Kennung) einlesen.

Wir gehen also in der Schleife wie folgt vor: Die ersten 4 Byte werden in einen String geladen. Dieser String wird nun mit der Zeichenkette »BMHD« verglichen. Stimmen beide Strings überein, so kann man davon ausgehen, daß es sich um einen BMHD-Chunk handelt. In diesem Chunk stehen alle Daten des Grafikformats, mit denen nun der Screen konfiguriert werden kann. Ist die Bildbreite größer als 320, handelt es sich in den meisten Fällen (doch nicht in allen, aber dazu am Schluß des GRAPHICS-Kapitels mehr) um ein Hires-Bild. Also setzen Sie das Flag HIRES im Feld viewModes Ihres Screens. Ähnliches gilt für die Bildhöhe: Ist diese größer als 256 (PAL), so ist es meist ein Interlace-Screen: Flag LACE im Feld viewModes setzen.

Die x- und y-Koordinaten des Bildes sind fast immer 0, womit das auch geklärt sein dürfte. Die Spezialmodi bei einer 6. Bitplane lassen wir mal außer acht.

Nun kommt ein Byte, das die Art des Masking angibt. Eine Maske ist im Prinzip eine Bitplane mit genau den gleichen Auflösungen wie die des eigentlichen Bildes. Diese Maske liegt über der sichtbaren Grafik. Steht an einer Stelle in der Maske eine 1, so ist das Bild an dieser Stelle sichtbar. Eine 0 gibt folglich an, daß das Bild an dieser Stelle nicht sichtbar ist.

Die »Stencil«-Funktion des Programms DPaint II ist ein Paradebeispiel für das Masking. Wie dem auch sei, ist das Masking-Byte des BMHD-Chunks 0, so wird, wie Sie sicher schon vermutet haben, kein Masking benutzt. Eine 1 heißt, daß die Maske dem BODY-Chunk angehängt wird. Steht hier eine 2, so bedeutet das, daß eine völlig durch-

sichtige Maske in einer bestimmten Farbe benutzt wird. Diese Farbe steht nach dem unbenutzten Byte und gibt an, welches Farbregister durchsichtig sein soll. Üblicherweise (d.h. ohne Masking) steht hier eine 0, da im Normalfall die Hintergrundfarbe durchsichtig ist.

Bleibt noch die letzte Möglichkeit, die das Masking-Byte annehmen kann: das »Lassoing«. Bei dieser Methode wird ein Rand mit der Breite eines Bildpunktes um die Grafik gezogen und nach innen gefüllt. Stößt der ausfüllende »Farbschwall« auf eine andere Farbe, so ist dieser Punkt (also der in der anderen Farbe) transparent und zeigt den Hintergrund, das eigentliche Bild also. Meines Wissens gibt es noch kein Amiga-Programm, das diese Technik ausnutzt, doch es scheint, als ob man damit sehr interessante Effekte erzielen kann.

Das nächste Byte gibt den Kompressionstyp an: Entweder keine Kompression (0) oder ByteRun-1-Kompression (1), die von DPaint II benutzt wird und später erklärt werden soll. Das unbenutzte Byte ist wenig interessant und die transparente Farbe wurde schon besprochen. Die nächsten beiden Bytes geben das Verhältnis Pixelbreite/Pixelhöhe an, was aber relativ uninteressant sein dürfte.

Die letzten beiden Words dieses Chunks geben die eigentliche Bildschirmgröße an, die aber fast immer identisch mit den ersten beiden Words des BMHDs ist.

Weiter geht's mit dem nächsten Chunk: er wird auf »CMAP« überprüft. Ist es ein CMAP-Chunk, so ist auch dessen Länge von Bedeutung: Teilen wir sie durch 3 erhalten wir die Anzahl der Benutzten Farbregister, da sich bekanntlich jede Farbe aus drei Komponenten zusammensetzt. So müssen wir dann eine zweite Schleife eröffnen, die einzelnen Komponenten laden und dann die Farbe mit SETRGB4 setzen, wobei zu beachten ist, daß die drei Komponenten erst durch 16 geteilt werden müssen, bevor sie für SETRGB4 verwendbar sind.

Irgendwann ist das Programm auch mit dem Einlesen der Farben fertig. Dann sollten wir uns an den größten Brocken wagen, den BODY-Chunk (CCRT lassen wir vorerst aus). Da dessen Länge uninteressant ist, kommt sie in den »Abfalleimer«, den wir am Programmanfang alloziert haben. Da die Bitplane-Daten zeilenweise angelegt sind, müssen wir hier wieder zwei Schleifen definieren, in denen die Daten direkt in die Bitplanes des Screens geschrieben werden, z.B.:

```
FOR i:=Ø TO Screenhöhe-1 DO
  FOR j:=Ø TO Screentiefe-1 DO
    Read(MeineDatei, Screen^.rastPort.bitMap.planes[j]+(Screenbreite/8*i), Screen-
breite/8);
  END;
END;
```

Nun sind Sie eigentlich in der Lage, einen einfachen IFF-Loader zu schreiben. Ich habe für diese Stelle ein Beispielprogramm geschrieben, das ein Bild angegebenen Namens einliest und anzeigt, Kompressierung, Spezialmodi und »Color-Cycling« wurden nicht berücksichtigt. Sie sollten auch darauf achten, daß es sich bei der angegebenen Datei wirklich um ein IFF-Bild (ILBM) handelt, sonst steckt das Programm in einer Endlosschleife fest. Auf der Programmdiskette befindet sich bereits ein einfaches IFF-Bild mit dem Namen »IFFBild«.

Doch jetzt zum Demoprogramm:

```
MODULE ShowIFF;

FROM SYSTEM     IMPORT ADR,ADDRESS;
FROM Intuition  IMPORT NewScreen,ScreenPtr,OpenScreen,CloseScreen,customScreen;
FROM Graphics   IMPORT RastPortPtr,BitMapPtr,SetRGB4,ViewModes,ViewModeSet;
FROM Dos        IMPORT Open,Close,Read,Write,FileHandlePtr,Delay,Exit;
FROM Exec       IMPORT AllocMem,FreeMem,MemReqs,MemReqSet;
FROM Arguments  IMPORT GetArg;
FROM Terminal   IMPORT WriteString,WriteLn;
FROM Strings    IMPORT Compare;

CONST
    dppv = "DPPV";
    bmhd = "BMHD";
    cmap = "CMAP";
    crng = "CRNG";
    body = "BODY";
    camg = "CAMG";
    ccrt = "CCRT";

VAR MeinScreen : ScreenPtr;
    ScreenDaten: NewScreen;
    RP         : RastPortPtr;
    BMap       : BitMapPtr;
    Datei      : FileHandlePtr;
    i,j        : INTEGER;
    a          : LONGINT;
    R,G,B      : [Ø..255];
    Abfall     : ADDRESS;
    Ende       : BOOLEAN;
    Use        : LONGINT;
    text       : ARRAY[Ø..3] OF CHAR;
    Buffer     : ARRAY[Ø..5] OF ADDRESS;
    Dateiname  : ARRAY[Ø..5Ø] OF CHAR;
```

```
BEGIN
  GetArg(1,Dateiname,i); (* Argument (Dateinamen) holen *)
   (* Screen-"Gerüst" erzeugen, das später ausgefüllt wird *)
  WITH ScreenDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=Ø; height:=Ø; depth:=Ø;
    detailPen:=Ø; blockPen:=1; viewModes:=ViewModeSet{};
    type:=customScreen;
    font:=NIL; gadgets:=NIL; defaultTitle:=NIL; customBitMap:=NIL;
  END;

  Datei:=Open(ADR(Dateiname),1ØØ5); (* Datei im "modeOldfile" öffnen *)
  IF Datei=Ø THEN
    WriteLn; WriteString("Keine solche Datei gefunden!");
    Exit(1ØØØ);
  END;

  Abfall:=AllocMem(1Ø8,MemReqSet{chip,memClear}); (* 1Ø8 Byte für uninteressante
  Daten allozieren *)
  Ende:=FALSE;

  Use:=Read(Datei,Abfall,12); (* Datei-Header überlesen *)

  WHILE Ende=FALSE DO
    Use:=Read(Datei,ADR(text),4);                  (* Kennung laden *)

    Use:=Compare(text,Ø,4,bmhd,TRUE);       (* BMHD? *)
    IF Use=Ø THEN              (* jawohl! *)
      Use:=Read(Datei,Abfall,4);            (* Schund überlesen *)
      Use:=Read(Datei,ADR(ScreenDaten.width),2);  (* Breite einlesen *)
      Use:=Read(Datei,ADR(ScreenDaten.height),2); (* Höhe einlesen *)
      Use:=Read(Datei,Abfall,3);      (* 3 Byte überlesen *)
      Use:=Read(Datei,ADR(ScreenDaten.depth),2);  (* Anzahl Planes einlesen *)
      Use:=Read(Datei,Abfall,11);     (* 11 Byte überlesen *)
      IF ScreenDaten.width > 32Ø THEN             (* Bild breiter als 32Ø? *)
        INCL(ScreenDaten.viewModes,hires);    (* muß HIRES sein! *)
      END;
      IF ScreenDaten.height > 256 THEN     (* Bild höher als 256? *)
        INCL(ScreenDaten.viewModes,lace);    (* muß LACE sein! *)
      END;
      MeinScreen:=OpenScreen(ScreenDaten);          (* Screen öffnen *)
      RP:=ADR(MeinScreen^.rastPort);     (* RastPort definieren *)
      BMap:=RP^.bitMap;       (* BitMap definieren *)
      FOR i:=Ø TO ScreenDaten.depth-1 DO    (* Zwischenspeicher für die *)
       Buffer[i]:=BMap^.planes[i];     (* Adresse der Bitplanes *)
      END;
    END;
```

```
  Use:=Compare(text,Ø,4,cmap,TRUE);                (* ist Kennung CMAP? *)
  IF Use=Ø THEN      (* ja! *)
    Use:=Read(Datei,ADR(a),4);     (* Anzahl Farben lesen *)
    a:=a DIV 3;        (* durch 3 teilen *)
    FOR i:=Ø TO a-1 DO
      Use:=Read(Datei,ADR(R),1);     (* Rotkomponente einlesen *)
      Use:=Read(Datei,ADR(G),1);     (* Grünkomponente einlesen *)
      Use:=Read(Datei,ADR(B),1);     (* Blaukomponente einlesen *)
      SetRGB4(ADR(MeinScreen^.viewPort),
      i,R DIV 16,G DIV 16,B DIV 16); (* Farben setzen! *)
    END;
  END;

  Use:=Compare(text,Ø,4,body,TRUE);      (* ist Kennung BODY? *)
  IF Use=Ø THEN        (* ja! *)
    Use:=Read(Datei,Abfall,4);     (* Abfall überlesen! *)
    FOR i:=Ø TO ScreenDaten.height-1 DO     (* Bitmap einlesen" *)
      FOR j:=Ø TO ScreenDaten.depth-1 DO
        Use:=Read(Datei,Buffer[j]+(ScreenDaten.width DIV 8 * i),
                ScreenDaten.width DIV 8);
      END;
    END;
    Ende:=TRUE;
  END;

  Use:=Compare(text,Ø,4,crng,TRUE);                (* CRNG? *)
  IF Use=Ø THEN        (* uninteressant! *)
    Use:=Read(Datei,Abfall,12);
  END;

  Use:=Compare(text,Ø,4,dppv,TRUE);                (* DPPV? *)
  IF Use=Ø THEN        (* uninteressant! *)
    Use:=Read(Datei,Abfall,1Ø8);
  END;

  Use:=Compare(text,Ø,4,camg,TRUE);                (* CAMG? *)
  IF Use=Ø THEN        (* uninteressant! *)
    Use:=Read(Datei,Abfall,8);
  END;

  Use:=Compare(text,Ø,4,ccrt,TRUE);                (* CCRT? *)
  IF Use=Ø THEN        (* uninteressant! *)
    Use:=Read(Datei,Abfall,18);
  END;
END; (* von WHILE *)
```

```
  Close(Datei);         (* Schließen *)
  Delay(5ØØ);         (* Warten *)
  CloseScreen(MeinScreen);
  FreeMem(Abfall,1Ø8);
END ShowIFF.
```

4.11.1 IFF – zweiter Streich

So, nun wissen Sie, wie man einen einfachen IFF-Lader schreibt. Was ist aber mit Dingen wie HAM, EHB, komprimierte Bilder und »ColorCycling« (mit dem tolle Effekte möglich sind)? Dafür ist dieses Kapitel da, das auf die Feinheiten des IFF/ILBM-Standards zu sprechen kommt.

Um die Spezialmodi zu erkennen, kommt jetzt auch der CAMG-Chunk zum Zuge. In ihm stehen, wie oben schon angedeutet, die ViewPort-Modi der Grafik in Zahlenform, wie sie die C- und Assembler-Programmierer benutzen, denn jedem Wert aus der Menge ViewPortSet ist intern eine Zahl zugeordnet, z.B. 0FH (# 15) für CUSTOMSCREEN. Hier eine Tabelle der für uns interessanten Werte:

Wert (hexadezimal)	Modus
0004H	Interlace (wird zu 0800H oder 0080H dazuaddiert, wenn es sich um ein Interlace-Bild handelt)
0800H	HAM
0080H	Halfbrite
6800H	(bei manchen HAM-Bildern)

Wir müssen jetzt nur den Wert einlesen und mit den Zahlen der Tabelle vergleichen, um den HAM- oder Halfbrite-Modus zu setzen.

Nun ist das Einlesen komprimierter Bilder an der Reihe. Handelt es sich um den Byte-Run-1-Algorithmus (das Kompressionsbyte im BMHD-Chunk hat den Wert 1), so ist der BODY-Chunk »gepackt«. Das erste (vorzeichenbehaftete) Byte eines Blocks in einer BODY-Zeile, nennen wir es x, gibt den Zustand der nächsten Bytes an. Zu vorzeichenbehafteten Bytes ist noch zu sagen, daß, wenn so ein Byte größer als 127 ist, die Differenz zu 256 die negative Zahl bildet. 128 wäre vorzeichenbehaftet also –128, 217 wäre –39, aber nun zurück zur Komprimierung von Grafiken. Ist x größer oder gleich 0, werden die folgenden x+1 Bytes ohne Änderung übernommen. Ist x jedoch kleiner als 0, wird das folgende Byte –x+1mal wiederholt. Diese Methode ist unheimlich effektiv, wenn man bedenkt, daß eine Leerzeile eines Bildes (oder eine Zeile ohne Farbänderung) in niedriger Auflösung normalerweise 40 Byte (8*40=320) schluckt, nach dem Komprimieren aber nur noch 2 ($D9 und $00, denn $D9 entspricht vorzeichenbehaftet –39 und – (–39)+1=40, womit wir wieder bei der Ursprungsgröße wären).

Noch ein Beispiel: 80 Punkte einer Zeile sind schwarz. In einer ungepackten IFF-Grafik stünde die Byte-Folge:

```
-ØØ -ØØ -ØØ -ØØ -ØØ -ØØ -ØØ -ØØ -ØØ -ØØ
```

So steht dort aber nur noch

```
-F7 -ØØ
```

Das hieße, daß wir acht Byte gespart hätten. Machen wir die Probe: 245 (0F5H) entspricht vorzeichenbehaftet –10. In unsere Formel eingesetzt kommt heraus:

```
-(-9)+1 = 1Ø Byte entspricht 1Ø*8 = 8Ø Punkte.
```

Das dürfte soweit geklärt sein.

Jetzt soll noch das »Color-Cycling« besprochen werden. Da das System keine Cycling-Routinen unterstützt, werden wir nicht umherkommen, selbst welche zu schreiben, und zwar eine für das Vorwärts- und eine für das Rückwärtsscrolling. Alle wichtigen Werte stehen im CCRT-Chunk und müssen nur noch verwertet werden. Das Scrollen der Farben erreicht man, indem man die Farbtabelle des ViewPorts bzw. deren Inhalt verschiebt.

4.12 Layers

Layers (zu deutsch Schicht) sind die Grundlagen für Windows, die auf ihnen basieren. Die meiste Arbeit, die durch den Umgang mit Fenstern anfällt, erledigen sie. Spezielle Befehle für die Programmierung von Layers sind in der »layers.library« angelegt, die vom M2-Compiler automatisch geöffnet und geschlossen wird.

Ein Layer kreiert man mit dem Befehl CREATEUPFRONTLAYER aus dem Layers-Definitionsmodul:

MeinLayer := CreateUpfrontLayer (MeinLayerInfo , MeineBitMap , x1 , y1 , x2 , y2, Flags , SuperBitMap);

MeinLayerInfo	LayerInfoPtr auf den LayerInfo-Record des Screens, auf dem das Layer erzeugt werden soll, der ausgefüllt werden soll (der letzte Nebensatz bezieht sich auf den LayerInfo-Record).
MeineBitMap	BitMapPtr auf die Bitmap, in die das Layer geschrieben werden soll.
x1	LONGINT-x-Koordinate der linken, oberen Ecke des Layers.
y1	LONGINT-y-Koordinate der linken, oberen Ecke des Layers.

x2	LONGINT-x-Koordinate der rechten, unteren Ecke des Layers.
y2	LONGINT-y-Koordinate der rechten, unteren Ecke des Layers.
Flags	LONGSET-Flags des Layers. Folgende Werte (aus LayersFlagSet) sind zulässig: layerSimple: Der Hintergrund des Layers wird nicht gesichert (wie bei SIMPLEREFRESH). layerSmart: Der Hintergrund des Layers wird gesichert. layerSuper: Das Layer ist ein Super-Bitmap-Layer, was wir noch besprechen werden. lf3: Reserviert. layerUpdating: ??? lf5: Reserviert. layerBackdrop: Das Layer ist hinter allen anderen Layers sichtbar (vgl. BACKDROP-Window). layerRefresh: ?? layerClipRectsLost: GRAPHICS setzt dieses Flag, wenn während eines Layer-Updates der Speicherplatz nicht mehr ausreicht.

Da aber der Typ LONGSET verlangt wird, müssen Sie für jeden Layertyp ein Bit setzen (nur die wichtigsten sind aufgeführt):

gesetztes Bit 0: LAYERSIMPLE
gesetztes Bit 1: LAYERSMART
gesetztes Bit 2: LAYERSUPER
gesetztes Bit 6: LAYERBACKDROP
gesetztes Bit 7: LAYERREFRESH

SuperBitMap	Dieser Parameter ist nur für den Super-Bitmap-Modus interessant und soll später besprochen werden.
MeinLayer	LayerPtr auf den Layer-Record des generierten Layers.

Das Layer erscheint dann, verlief alles fehlerfrei, vor den anderen Layers, vorrausgesetzt, Sie wählten nicht das Flag LAYERBACKDROP.

Bei dem Parameter »MeinLayerInfo« wurde der LayerInfo-Record angesprochen, aber wie kommt man an diesen heran? Er steht im Feld layerInfo des Screens. Man muß also

nur seine Adresse angeben, oder einen neuen LayerInfo-Record mit der Funktion NEWLAYERINFO ausfüllen:

MeinLayerInfo := NewLayerInfo();

MeinLayerInfo	LayerInfoPtr auf den gerade ausgefüllten LayerInfo-Record.

Der LayerInfo-Record birgt eigentlich keine sonderlich verständlichen oder interessanten Daten für den Programmierer, so daß auf eine Besprechung verzichtet wird.

Zum Layer-Record: Dieser ist in etwa vergleichbar mit dem Window-Record:

```
Layer=RECORD
  front              : LayerPtr           ; Zeiger auf das Layer im Vordergrund
  back               : LayerPtr           ; Zeiger auf das Layer im Hintergrund
  clipRect           : ClipRectPtr        ; Zeiger auf den ersten ClipRect-Record
  rp                 : RastPortPtr        ; Zeiger auf den RastPort des Layers
  bounds             : Rectangle          ; Ausmaße des Layers
  reserved           : ARRAY[Ø..3] OF BYTE ; reserviert
  priority           : CARDINAL           ; für den Systemgebrauch
  flags              : CARDINAL           ; Flags des Layers
  superBitMap        : BitMapPtr          ; Zeiger auf evt. Super-Bitmap
  superClipRect      : ClipRectPtr        ; Zeiger auf den ersten ClipRect bei
                                            einer Super-Bitmap
  window             : ADDRESS            ; Adresse des Windows
  scrollX            : INTEGER            ; x-Position des Layers bei
                                            Super-Bitmaps
  scrollY            : INTEGER            ; y-Position des Layers bei
                                            Super-Bitmaps
  cr                 : ClipRectPtr        ; fürs System
  cr2                : ClipRectPtr        ; fürs System
  crnew              : ClipRectPtr        ; fürs System
  superSaveClipRects: ClipRectPtr         ; fürs System
  cliprects          : ClipRectPtr        ; fürs System
  layerInfo          : LayerInfoPtr       ; Zeiger auf LayerInfo des Layers
  lock               : SignalSemaphore    ; fürs System
  reserved3          : ARRAY[Ø..7] OF BYTE ; reserviert
  clipRegion         : RegionPtr          ; fürs System
  saveClipRects      : RegionPtr          ; fürs System
  reserved2          : ARRAY[Ø..21] OF BYTE; reserviert
  damageList         : RegionPtr          ; reserviert
END;
```

clipRect	Ein ClipRect legt jeweils einen (rechteckigen) Ausschnitt eines Layers fest:

```
ClipRect=RECORD
  next     : ClipRectPtr ; Zeiger auf nächstes ClipRect
  prev     : ClipRectPtr ; Zeiger auf vorheriges ClipRect
  lobs     : LayerPtr    ; Zeiger auf das Layer des ClipRects
  bitMap   : BitMapPtr   ; Zeiger auf die Bitmap des Layers
  bounds   : Rectangle   ; Ausmaße des ClipRects
  p1       : ClipRectPtr ; fürs System
  p2       : ClipRectPtr ; fürs System
  reserved: LONGINT      ; reserviert
  flags    : LONGINT     ; fürs System
END;
```

bounds — In diesem Feld werden die Ausmaße des Layers festgelegt. Werden diese Grenzen überschritten, wird der Rest der Zeichnung abgeschnitten (»geclippt«). Ein Rectangle-Record sieht so aus:

```
Rectangle=RECORD
  minX: INTEGER ; x-Koordinate der linken, oberen Ecke des Layers
  minY: INTEGER ; y-Koordinate der linken, oberen Ecke des Layers
  maxX: INTEGER ; x-Koordinate der rechten, unteren Ecke des Layers
  maxY: INTEGER ; y-Koordinate der rechten, unteren Ecke des Layers
END;
```

flags — Hier stehen die gewählten Layer-Flags als Bitwerte:

```
gesetztes Bit Ø: LAYERSIMPLE
gesetztes Bit 1: LAYERSMART
gesetztes Bit 2: LAYERSUPER
gesetztes Bit 6: LAYERBACKDROP
gesetztes Bit 7: LAYERREFRESH
```

superBitMap — Haben Sie den Super-Bitmap-Modus gewählt (dazu später), so muß dieses Feld auf die Super-Bitmap zeigen.

superClipRect — Für dieses Feld gilt eigentlich das gleiche wie für das Feld clipRect, nur daß sich superClipRect auf Super-Bitmaps bezieht.

window — Wenn Sie Ihr Layer mit einem Intuition-Window zusammenarbeiten lassen, so können Sie hier die Adresse des Windows eintragen, was bei Super-Bitmap-Layers ganz nützlich ist, wie Sie später noch sehen werden.

Nun können Sie mit den Grafikbefehlen Ihre Layers vollzeichnen, bis sie überlaufen, wenn Sie als RastPort den des betreffenden Layers angeben.

Soll Ihr Layer wieder vom Bildschirm verschwinden, benuzen Sie einfach das Kommando DELETELAYER:

ok := DeleteLayer (MeinLayer);

MeinLayer	LayerPtr auf das zu löschende Layer.
ok	LONGINT-Rückgabewert, der entweder 0 (das Löschen des Layers verlief fehlerfrei) oder ein anderer Wert ist (es trat bei der Ausführung dieses Befehls irgendein Fehler auf).

Es gibt auch noch ein paar Layerspezifische Befehle, die zuweilen ganz nützlich sein können, z.B. MOVELAYER, der das angegebene Layer bewegt:

ok := MoveLayer (MeinLayer , dx , dy);

MeinLayer	LayerPtr auf das zu bewegende Layer.
dx	LONGINT-x-Entfernung, die das Layer bei der Bewegung zurücklegen soll. Ein negativer Wert bewegt das Layer nach links.
dy	LONGINT-y-Entfernung, die das Layer bei der Bewegung zurücklegen soll. Wird hier ein negativer Wert angegeben, wird das Layer nach oben bewegt.
ok	LONGINT-Rückgabewert, der 0 ist, wenn kein Fehler bei der Bewegung auftrat, oder 1, wenn etwas schiefging.

Um die Größe eines Layers zu ändern, ist SIZELAYER da:

ok := SizeLayer (MeinLayer , dx , dy);

MeinLayer	LayerPtr auf das Layer, dessen Größe geändert werden soll.
dx	LONGINT-Wert, der angibt, um wieviele Bildpunkte das Layer in x-Richtung vergrößert werden soll. Ein negativer Wert bewirkt an dieser Stelle eine Verkleinerung.
dy	LONGINT-y-Wert, der angibt, um wieviele Bildpunkte das Layer in y-Richtung vergrößert werden soll. Geben Sie hier einen negativen Wert an, so wird das Layer verkleinert.

Auch die Prioritäten der Layers unter sich können geändert werden. MOVELAYER-INFRONTOF beispielsweise bewegt ein Layer vor ein anderes:

ok := MoveLayerInFrontOf (MeinLayer , ZielLayer);

MeinLayer	LayerPtr auf das Layer, das vor einem anderen erscheinen soll.
ZielLayer	LayerPtr auf das Layer, vor das »MeinLayer« bewegt werden soll.
ok	LONGINT-Rückgabewert, der entweder 0 (alles klar) oder 1 (ein Fehler trat auf) ist.

Für das Gegenteil ist BEHINDLAYER nutze, das ein Layer hinter alle anderen schiebt (also quasi zum BACKDROP-Layer macht):

ok := BehindLayer (MeinLayer);

MeinLayer	LayerPtr auf das Layer, das hinter allen anderen erscheinen soll.
ok	LONGINT-Rückgabewert, der entweder (alles klar) oder 1 (ein Fehler trat auf) ist.

Nun ist es Zeit für das erste Beispielprogramm zum Thema Layers:

```
MODULE LayerDemo;

FROM SYSTEM     IMPORT ADR,LONGSET;
FROM Intuition  IMPORT OpenWindow,CloseWindow,NewWindow,WindowPtr,WindowFlags,
                       WindowFlagSet,IDCMPFlags,IDCMPFlagSet,ScreenFlags,
                       ScreenFlagSet;
FROM Graphics   IMPORT LayerPtr,RastPortPtr,LayerInfoPtr,BitMapPtr,Text,Move,
                       LayerFlags,LayerFlagSet,Draw;
FROM Layers     IMPORT CreateUpfrontLayer,DeleteLayer,MoveLayer;
FROM Dos        IMPORT Delay;

VAR WindowDaten: NewWindow;
    MeinWindow : WindowPtr;
    MeinInfo   : LayerInfoPtr;
    MeinLayer  : LayerPtr;
    RP         : RastPortPtr;
    MeineBMap  : BitMapPtr;
    ok,i       : LONGINT;

BEGIN
  WITH WindowDaten DO
    leftEdge:=0; topEdge:=0; width:=100; height:=20; detailPen:=0;
    blockPen:=1;
    idcmpFlags:=IDCMPFlagSet{}; flags:=WindowFlagSet{windowDrag,activate,
    windowDepth}; firstGadget:=NIL; checkMark:=NIL; screen:=NIL;
```

```
    bitMap:=NIL;
    title:=ADR("LAYERS"); type:=ScreenFlagSet{wbenchScreen};
  END;

  MeinWindow:=OpenWindow(WindowDaten);

  MeineBMap:=MeinWindow^.wScreen^.rastPort.bitMap; (* Bitmap merken *)
  MeinInfo:=ADR(MeinWindow^.wScreen^.layerInfo);   (* LayerInfo merken *)

   MeinLayer:=CreateUpfrontLayer(MeinInfo,MeineBMap,0,0,540,50,LONGSET{1},NIL);
   RP:=MeinLayer^.rp;         (* RastPort des Layers *)

  (* Ab hier wird ein bißchen in das Layer gezeichnet, damit man es etwas
     deutlicher sieht. *)
  Move(RP,0,0); Draw(RP,540,0); Draw(RP,540,50); Draw(RP,0,50);
  Draw(RP,0,0);
  Move(RP,10,10);
  Text(RP,ADR("Dies ist ein LAYERSMART-Layer."),19);
  Move(RP,10,20);
  Text(RP,ADR("Erinnert Sie an ein Window, oder?"),34);

  FOR i:=0 TO 50 DO        (* Layer bewegen *)
    ok:=MoveLayer(MeinLayer,1,2);
  END;

  Delay(500);

  ok:=DeleteLayer(MeinLayer);        (* Weg mit dem Layer! *)

  CloseWindow(MeinWindow);
END LayerDemo.
```

4.12.1 Super-Bitmap-Layers

Der Super-Bitmap-Layer ist eine Unterart des Layers, die es in sich hat: Sie können eine übergroße Bitmap (maximal 1024*1024 Bildpunkte!) definieren und Ihr Layer wie eine Lupe darüberscrollen lassen. Die Verwirklichung dieses Problems ist eigentlich gar nicht schwer, da sie voll vom System unterstützt wird. Man muß nur eine eigene Bitmap mit den gewünschten Ausmaßen (1024*1024 Punkte, z.B.) und die entsprechende Anzahl an Bitplanes definieren. Dazu benutzt man am besten die Befehle INITBIT-MAP und ALLOCRASTER. Dann definieren Sie Ihren eigenen Screen und ein Window und öffnen diese, wobei Sie beachten sollten, daß es sich um ganz normale Screens und Windows handeln muß (ohne CUSTOMBITMAP oder ähnliche Dinge). Darauf müssen Sie ein Layer mit CREATEUPFRONTLAYER öffnen, dessen Abmessungen genau mit denen des Fenster-Layers übereinstimmen sollten, damit keiner bemerkt, daß es sich um ein Window und ein seperates Layer handelt. Wichtig ist noch, daß Sie als Layer-Flag das Bit für LAYERSUPER (Bit 2) angeben und einen Zeiger auf

Ihre Super-Bitmap übergeben. Sie können nun unter Angabe des Layer-RastPorts munter in die Super Bitmap zeichnen und so eine schöne Zeichenfläche, über die das Layer scrollen soll, definieren. Scrollen kann man das Layer mit einem entsprechenden Befehl aus dem Layers-Definitionsmodul:

ScrollLayer (MeinLayer , dx , dy);

MeinLayer	LayerPtr auf das zu scrollende Layer.
dx	LONGINT-Wert, der angibt, um wieviele Bildpunkte das Layer in x-Richtung gescrollt werden soll. Die Angabe eines negativen Wertes hat zur Folge, daß das Layer nach links gescrollt wird.
dy	LONGINT-Wert, der angibt, um wieviele Bildpunkte das Layer in y-Richtung gescrollt werden soll. Geben Sie hier einen negativen Wert an, wird das Layer nach oben gescrollt.

Das wäre ja alles schön und gut, wenn da nicht noch ein Haken wäre: klickt der Benutzer mit der Maus Ihr Layer an (Benutzer haben oft die Eigenschaft, mit der Maus irgendwo herumzuklicken), wird das darunterliegende Fenster deaktiviert und kann schlecht wieder aktiviert werden, weil ja das Layer darüberliegt. Ist das Fenster einmal deaktiviert, kann es keine IDCMP-Angaben mehr empfangen, was aber oft ungewollt ist, denn es wäre ja ganz schön, wenn das Programm auf Mausklick endet, daß evt. vorhandene Menüs oder Gadgets abgefragt werden können usw. Um diesem Problem abzuhelfen, ist ein kleiner Trick vonnöten: Wir drehen dem Window einfach unser Layer an. Das wird ganz einfach dadurch erreicht, indem wir das Feld window des Layer-Records mit der Adresse unseres Fensters füllen.

Hier noch ein kleiner Tip: Wie bei den Bobs vermindert auch beim Scrollen von Layers der WAITTOF-Befehl das Flackern ungemein, wenn er es auch nicht ganz aufhebt.

Mit einem weiteren Beispielprogramm schließen wir auch das Thema Layers ab:

```
MODULE SuperLayerDemo;

FROM SYSTEM     IMPORT ADR,ADDRESS,INLINE,LONGSET;
FROM Intuition IMPORT OpenWindow,CloseWindow,NewWindow,WindowPtr,WindowFlags,
                      WindowFlagSet,IDCMPFlags,IDCMPFlagSet,OpenScreen,
                      CloseScreen,ScreenPtr,customScreen,NewScreen;
FROM Graphics   IMPORT LayerPtr,RastPortPtr,LayerInfoPtr,BitMapPtr,Text,SetAPen,
                      LayerFlags,LayerFlagSet,Draw,InitBitMap,AllocRaster,Move,
                      ViewModes,ViewModeSet,FreeRaster,BitMap,LoadRGB4,SetRast,
                      RectFill,WaitTOF;
```

```
FROM Layers     IMPORT CreateUpfrontLayer,DeleteLayer,ScrollLayer;
FROM Dos        IMPORT Delay;

VAR WindowDaten     : NewWindow;
    ScreenDaten     : NewScreen;
    MeinWindow      : WindowPtr;
    MeinScreen      : ScreenPtr;
    MeinInfo        : LayerInfoPtr;
    MeinLayer       : LayerPtr;
    RP              : RastPortPtr;
    MeineBMap       : BitMap;
    ScreenBMap      : BitMapPtr;
    ok,i,a,b,c,d,x,y: LONGINT;

PROCEDURE Farben; (* $E- *)
  BEGIN
    INLINE(0000H,0FFFH,0F00H,000FH,00F0H,0FB0H,0C0CH,0999H);
  END Farben;

BEGIN
  WITH ScreenDaten DO
    leftEdge:=0; topEdge:=0; width:=320; height:=256; depth:=3;
    detailPen:=0; blockPen:=1; viewModes:=ViewModeSet{}; font:=NIL;
    defaultTitle:=ADR("SUPER-LAYER-DEMO !"); type:=customScreen;
    customBitMap:=NIL; gadgets:=NIL;
  END;

  WITH WindowDaten DO
    leftEdge:=0; topEdge:=0; width:=311; height:=230; detailPen:=0;
    blockPen:=1;
    idcmpFlags:=IDCMPFlagSet{}; flags:=WindowFlagSet{windowDrag,activate,
    windowDepth}; firstGadget:=NIL; checkMark:=NIL; screen:=NIL;
    bitMap:=NIL;
    title:=ADR("LAYERS"); type:=customScreen;
  END;

  MeinScreen:=OpenScreen(ScreenDaten);
  WindowDaten.screen:=MeinScreen;
  MeinWindow:=OpenWindow(WindowDaten);

  LoadRGB4(ADR(MeinScreen^.viewPort),ADR(Farben),8);

  InitBitMap(MeineBMap,3,1024,512);            (* Bitmap initialisieren *)
  FOR i:=0 TO 2 DO    (* Planes bereitstellen *)
    MeineBMap.planes[i]:=AllocRaster(1024,512);
  END;
```

```
MeinInfo:=ADR(MeinScreen^.layerInfo);  (* LayerInfo holen *)
ScreenBMap:=ADR(MeinScreen^.bitMap);  (* alte Bitmap holen *)

(* Grenzen des Window-Layers holen *)
a:=MeinWindow^.wLayer^.bounds.minX; b:=MeinWindow^.wLayer^.bounds.minY;
c:=MeinWindow^.wLayer^.bounds.maxX; d:=MeinWindow^.wLayer^.bounds.maxY;
(* neuen Layer kreieren (Type LAYERSUPER) *)

MeinLayer:=CreateUpfrontLayer(MeinInfo,ScreenBMap,a,b,c,d, LONGSET{2},
           ADR(MeineBMap));
RP:=MeinLayer^.rp;          (* RastPort des Layers *)
SetRast(RP,Ø);              (* Layer löschen *)
SetAPen(RP,1);              (* Gittermuster zeichnen *)
FOR y:=Ø TO 511 BY 32 DO
  IF y>=64 THEN SetAPen(RP,y DIV 64); END;
  Move(RP,Ø,y);
  Draw(RP,1Ø23,y);
END;
SetAPen(RP,1);
FOR x:=Ø TO 1Ø23 BY 32 DO
  IF x>=128 THEN SetAPen(RP,x DIV 128); END;
  Move(RP,x,Ø);
  Draw(RP,x,511);
END;

SetAPen(RP,2);        (* Start- und Endemarken *)
RectFill(RP,Ø,Ø,64,16);      (* zeichnen *)
Move(RP,8,12);
Text(RP,ADR("START!"),6);
SetAPen(RP,3);
RectFill(RP,967,495,1Ø23,511);
Move(RP,975,5Ø7);
Text(RP,ADR("ENDE!"),5);

FOR x:=Ø TO 7Ø3 DO       (* horizontal scrollen *)
  ScrollLayer(MeinLayer,1,Ø);
  WaitTOF;
END;
FOR y:=Ø TO 281 DO       (* vertikal scrollen *)
  ScrollLayer(MeinLayer,Ø,1);
  WaitTOF;
END;
```

```
  Delay(100);

  ok:=DeleteLayer(MeinLayer);        (* Weg mit dem Layer! *)

  CloseWindow(MeinWindow);
  CloseScreen(MeinScreen);
  FOR i:=0 TO 2 DO          (* Planes freigeben *)
    FreeRaster(MeineBMap.planes[i],1024,512);
  END;
END SuperLayerDemo.
```

4.13 Tips & Tricks

Jetzt, da Sie mit dem Hauptinhalt des GRAPHICS-Kapitels durch sind, möchte ich Ihnen noch ein paar kleine, aber nützliche Tips präsentieren, die das Arbeiten mit dem GRAPHICS-System erleichtern.

4.13.1 Overscan

Ich habe Ihnen, seit ich angefangen habe über Screens zu schreiben, beigebracht, daß die höchste Auflösung eines Screens 640*512 Punkte ist. Jetzt kommt's: das stimmt nicht. Mit dem sogenannten Overscan-Modus ist es möglich, Screens zu definieren, die auch den letzten Rest des Bildschirms ausfüllen, so daß nicht die kleinste Spur von Rand bleibt. Dieser Modus wird eingeschaltet, indem die Werte in den Feldern dxOffset und dyOffset des View-Records geändert werden. Der View-Record muß natürlich zuerst mit VIEWADDRESS geholt werden und nach der Änderung wieder mit REMAKE-DISPLAY erneuert werden. Statt die Werte des Views zu ändern, können Sie Ihre Modifikation auch in den Preferences (z.B. mit dem Preferences-Programm) vornehmen, aber das wäre wohl zuviel Arbeit, wenn man nicht ständig im Overscan-Modus arbeiten möchte. Da die neuen Werte nicht nur für einen Screen, sondern für alle gelten, sollten am Programmende wieder die alten gesetzt werden. Nachdem Sie den View modifiziert haben, können Sie Screens in höheren Auflösungen, als sonst üblich, öffnen. Diese sind:

Screen-Modus	alte Auflösung	neue Auflösung
Lores	320*256 Punkte	352*290 Punkte
Lores+Interlace	320*512 Punkte	352*580 Punkte
Hires	640*256 Punkte	672*290 Punkte
Hires+Interlace	640*512 Punkte	672*580 Punkte

Dann gibt es noch die extreme Auflösung von 704*580 Punkten, die aber nur auf wirklich hervorragenden Monitoren, die z.T. teuerer sind als der Amiga selbst, mit absolut

hoher Auflösung ganz sichtbar ist. Generell genügen die in der Tabelle aufgeführten Auflösungen völlig. Aber auch die Nachteile des Overscan sollen nicht verschwiegen werden:

- In den Grenzbereichen zwischen alter und neuer Auflösung können keine Sprites dargestellt werden.
- An den Bildrändern treten Unschärfen auf.
- Der verbrauchte Speicherplatz ist gerade bei sehr speicherintensiven Screens (HAM, Hires+Interlace mit 4 Planes, ...) ungleich höher.

Auch zum Overscan-Modus soll ein Beispielprogramm folgen, das die absolut extreme Auflösung von 704*580 Punkten demonstriert:

```
MODULE Overscan;

FROM SYSTEM     IMPORT ADR;
FROM Intuition  IMPORT NewScreen,ScreenPtr,OpenScreen,CloseScreen,customScreen,
                       ViewAddress,RemakeDisplay;
FROM Graphics   IMPORT ViewModes,ViewModeSet,ViewPtr,Move,Draw,RastPortPtr;
FROM Dos        IMPORT Delay;

VAR MeinScreen : ScreenPtr;
    ScreenDaten: NewScreen;
    MeinView   : ViewPtr;
    RP         : RastPortPtr;
    i,j,x,y    : INTEGER;

BEGIN
  WITH ScreenDaten DO
    leftEdge:=0; topEdge:=0; width:=704; height:=560;
    depth:=2; detailPen:=0; blockPen:=1;
    viewModes:=ViewModeSet{hires,lace};
    type:=customScreen; font:=NIL; defaultTitle:=ADR("OVERSCAN-SCREEN");
    gadgets:=NIL; customBitMap:=NIL;
  END;

  MeinView:=ViewAddress();        (* Adresse des Views holen *)
  x:=MeinView^.dxOffset; y:=MeinView^.dyOffset;     (* alte Werte sichern *)
  MeinView^.dyOffset:=30; MeinView^.dxOffset:=112; (* Overscan einschalten *)
  RemakeDisplay;         (* Display erneuern *)
```

```
  MeinScreen:=OpenScreen(ScreenDaten);
  RP:=ADR(MeinScreen^.rastPort);

  FOR i:=0 TO 695 BY 40 DO          (* irgendwas zeichnen *)
    FOR j:=0 TO 555 BY 40 DO
      Move(RP,0,i); Draw(RP,j,0);
    END;
  END;

  Delay(250);

  MeinView^.dyOffset:=y; MeinView^.dxOffset:=x;      (* alte Werte setzen *)
  RemakeDisplay;          (* Display erneuern *)

  CloseScreen(MeinScreen);
END Overscan.
```

4.13.2 Interlace-Tips

Um das starke Flackern des Screens im Interlace-Modus zu lindern, gibt es einige Tips, von denen die wirkungsvollsten hier aufgelistet sind:

- Benutzen Sie Farbe mit geringer Intensität (dunkle Farben wie Dunkelgrau, Dunkelrot, Dunkelblau, ...).
- Ihre Farben sollten sehr kontrastarm sein. Vermeiden Sie Farbsprünge wie von Dunkelblau auf Weiß oder ein helles, intensives Gelb.
- Drehen Sie beim Betrachten von Interlace-Bildern den Kontrastregler an Ihrem Monitor herunter.
- Dieser Tip ist noch wirkungsvoller als der vorherige: Setzen Sie beim Betrachten von Interlace-Bildern eine gute bis sehr gute Sonnenbrille auf (am besten sind natürlich »Ray Ban«-Brillen, wenn sie auch sehr teuer sind).

Und jetzt noch ein Tip, den die Entwickler des Amiga zur Vermeidung des Interlace-Flackerns gaben:

- Schließen Sie Ihren Computer an einen multisynchronen Monitor an, und bauen Sie in den Amiga einen MicroWay FlickerFixer ein (NEC Multisync II, z.B.). Der Interlacemodus wird dann völlig flimmerfrei dargestellt. Über diesen Rat braucht man kaum Worte zu verlieren: welcher nicht-kommerzielle Benutzer schließt seinen Computer schon an einen 2000–3000 DM teuren Monitor mit Graphikkarte an?!

4.13.3 Mathematische Grafiken

Programmieren Sie mathematische Grafik, bei deren Berechnung Sie Fließkommazahlen benötigen (Funktionsplotter, Fraktale, Raytracer usw.), haben Sie die Qual der Wahl, welches Zahlenformat Sie nehmen sollen, entweder das schnelle Amiga-FFP-Format mit sechs Nachkommastellen oder das etwas langsamere REAL-(IEEE-)Format mit 20 Nachkommastellen. Bei mathematischen Grafiken sollten Sie auf jeden Fall das FFP-Format nehmen, da es dort ja meist auf Geschwindigkeit ankommt und die 20 Nachkommastellen des REAL-Formates sich sowieso nicht auf die Grafik auswirken.

4.13.4 Fading

Unter »Fading« versteht man das langsame Einblenden einer Grafik. Dieser Effekt wird aber nicht durch komplizierte Ein-/Überblendungsalgorithmen, die vielleicht sogar in Assembler geschrieben sein müssen, sondern vielmehr durch geschickte Color-Map-Handhabung erreicht. Damit meine ich das langsame Heraufsetzen jeder Farbkomponente von Schwarz (also 0000H) an, bis schließlich Farbe selbst erreicht ist. Programmieren Sie doch Ihr eigenes »Fading«!

4.14 Überblick

In folgendem Bild wird nochmal die Stellung der einzelnen Systemkomponenten von GRAPHICS deutlich:

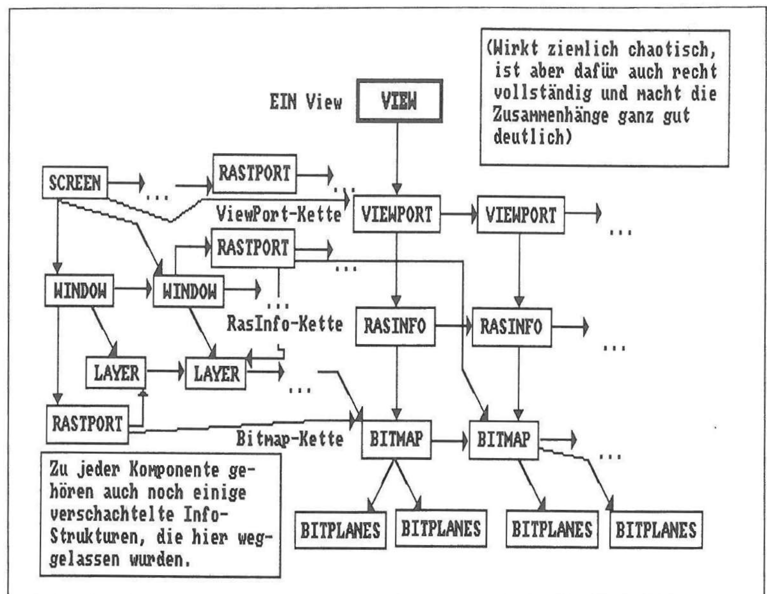


Bild 4.4: Die einzelnen Systemkomponenten von GRAPHICS

Kapitel 5

Der Copper

Der Copper ist, wie der Name schon andeutet, ein Coprozessor. Er wird so genannt, weil er über drei eigene Befehle verfügt, mit denen man (auch) auf die Hardware-Register zugreifen kann. Mit diesen drei Befehlen kann man, wie bei einem »richtigen« Prozessor, Programme schreiben, Copperlisten genannt. Natürlich haben diese Listen nicht die Komplexität echter Computerprogramme, mit drei Befehlen ist der Anwender ja auch in seinem Tun beschränkt, doch gerade was die Bildschirmsteuerung angeht, offenbart uns dieser Chip ungeahnte Möglichkeiten, denn mit ihm ist es möglich, den Elektronenstrahl, der alle 1/50 Sekunde von oben nach unten über den Bildschirm fährt und dabei das augenblicklich sichtbare Bild zeichnet, abzufragen. Außerdem läuft ein Copperprogramm im Interrupt, also parallel zum Hauptprogramm. Wenn das Betriebssystem nicht gerade den Copper benutzt, können wir uns seiner bedienen.

5.1 Die Programmierung des Coppers

Um den Copper zu programmieren, gibt es prinzipiell zwei Möglichkeiten: 1. Mittels direkter Ansteuerung in Maschinensprache und mühevollem Umrechnen einer Copperliste in einzelne Bytes oder 2. mittels dreier Befehle aus der Graphics-Library, die wir einfach nur aus GfxMacros zu importieren brauchen.

Sie sehen schon, der zweite Weg ist dem ersten vorzuziehen (es sei denn, Sie sind ein ausgesprochener Freak unübersichtlicher und umständlich erstellter Programme). Doch kommen wir nun zu den Copper-Befehlen:

MOVE	Er schreibt einen beliebigen Wert in ein beliebiges Hardware-Register.
WAIT	Dieser Befehl wartet, bis der Elektronenstrahl an einer bestimmten Bildschirmposition angelangt ist.
SKIP	Mit diesem Befehl kann man das nächste Kommando überspringen, wenn die angegebene Bedingung erfüllt ist.

Der Skip-Befehl ist das wohl am wenigsten gebrauchte Copper-Kommando. Da er auch unter den Graphics-Befehlen nicht vertreten ist, sei er hier nur formhalber erwähnt. Sollten Sie sich trotzdem für ihn interessieren, möchte ich Sie auf das »Hardware-Reference-Manual« oder auf »AMIGA-Intern« (beide im Literaturanhang erwähnt) verweisen.

5.2 Die Copper-Befehle

Um aber eine Copperliste erstellen zu können, muß erst genügend Platz reserviert und ein Zeiger auf dieselbe initialisiert werden. Der Zeiger muß vom Typ UCopListPtr, den wir aus Graphics importieren können, sein. Mit der Funktion ALLOCMEM aus Exec dürfte auch die Speicherreservierung kein Problem sein. Wichtig ist nur, daß der zu reservierende Speicherbereich CHIP-Memory, also die untersten 512 Kbyte, und bereits mit Nullbytes aufgefüllt sein muß. Um dies zu erreichen, importieren wir die Aufzählung MemReqs und die dazugehörige Menge MemReqSet aus Exec und schreiben

```
Copperliste:=AllocMem(SIZE(Copperliste^), MemReqSet{chip,memClear});
```

und unser UCopListPtr »Copperliste« zeigt auf den Bereich, in den unser Copperprogramm geschrieben wird. ALLOCMEM wird übrigens im Kapitel über Exec ausführlich beschrieben.

Wie oben schon angedeutet, verwenden wir für die Copperinstruktionen MOVE und WAIT die Befehle CMOVE und CWAIT aus GfxMacros. Deren Syntax lautet:

CMOVE(Copperliste, Register, Wert);

Copperliste	Ein Zeiger vom Typ UCopListPtr, den die Funktion ALLOCMEM zurückgibt.
Register	Ein Feld des Records Custom aus Hardware.
Wert	Ein INTEGER-Wert, der ins Register »Register« geschrieben werden soll.

CWAIT(Copperliste, yPos, xPos);

Copperliste	Ein Zeiger auf die Copperliste, siehe CMOVE.
yPos	Ein INTEGER-Wert, der y-Koordinate der Position des Rasterstrahls, auf die gewartet werden soll, angibt.
xPos	Ebenfalls ein INTEGER-Wert, der die x-Koordinate angibt

Sie sollten allerdings beachten, daß die y- und x-Koordinaten die Werte 263 und 223 nicht überschreiten dürfen, denn sonst treten mitunter recht eigentümliche Effekte auf, die mit Ihrer Copperliste nichts mehr zu tun haben. CWAIT bezieht sich glücklicherweise ausschließlich auf die Copperliste, nicht aber auf das Hauptprogramm, das ganz cool und unbehelligt weiterläuft.

Ferner dessen muß noch das Ende der Copperliste markiert werden. Man tut das normalerweise mit dem Befehl CEND, der sich wie die anderen beiden in GfxMacros befindet. Statt CEND(Copperliste,a,b) können Sie natürlich auch CWAIT(Copperliste, 10000, 256) schreiben, da dies eine für den Elektronenstrahl niemals erreichbare Position ist (zumindest zu dem Zeitpunkt der Erstellung dieses Buches, möglicherweise wird Commodore die Customchips irgendwann mal überarbeiten, aber welche Neuheiten dann kommen, ist noch nicht für die Öffentlichkeit zugänglich) und der Copper so die Instruktionsliste nicht weiter abarbeiten kann. Die Bedeutung der Parameter »a« und »b« ist mir allerdings unklar, ist doch im »Rom-Kernel-Manual«, Band I, der Befehl ohne Parameter definiert. Es ist wohl das sinnvollste, für »a« sowie »b« den Wert 0 anzugeben. So, nun müssen wir nur noch einen Zeiger auf den ViewPort (Typ ViewPortPtr, Import aus Graphics) des Screens, auf dem die Copperliste erscheinen soll deklarieren und in das Feld »uCopIns« den Zeiger auf unsere Instruktionsliste schreiben. Das hört sich komplizierter an, als es ist, es geschieht mit

```
VPort:=ADR(Screenzeiger^.viewPort);
VPort^.uCopIns:=Copperliste.
```

Wenn Sie bis jetzt durchgehalten haben, dann müssen Sie dem Amiga nur noch mitteilen, daß er Ihre Instruktionsliste auch ausführen soll. Dazu nehmen wir die parameterlose Prozedur RETHINKDISPLAY (Intuition-Import):

RethinkDisplay;

Wichtig ist nur noch, am Programmende den Screen, auf dem die Copperliste dargestellt wird, wieder zu schließen, den Rest erledigt Intuition automatisch. Sollte bis jetzt kein Programmierfehler aufgetreten sein, müßte die Copperliste jetzt abgearbeitet werden.

5.3 Programmierpraxis

Um die ganze graue Theorie zu veranschaulichen, soll nun ein Beispielprogramm kommen, das eine einfache Copperliste erstellt und initialisiert. Unser Programm soll einen einfachen Screen, dessen Hintergrundfarbe in der Bildschirmmitte von Blau auf Rot wechselt, darstellen. Damit wir ein solches Programm per Copper realisieren können, brauchen wir die Farbregister, ein Bestandteil der Hardwareregister. Sie sind ein Array-Feld des Records Custom aus dem Modul Hardware:

```
Custom = RECORD
  ...
  color: ARRAY [Ø..31] OF CARDINAL;
END;
```

Die einzelnen Farbregister sind zuständig für:

```
Register           Aufgabe
color[Ø]  Hintergrundfarbe   (Blau)
color[1]  Vordergrundfarbe 1 (Weiß)
color[2]  Vordergrundfarbe 2 (Schwarz)
color[3]  Vordergrundfarbe 3 (Orange)
...
color[31] Vordergrundfarbe 31
```

In Klammern stehen die Farben, die die Register beim normalen 4-Farb-Workbench-Screen enthalten (mit dem Preferences-Programm zu ändern). Der Wert ist ein hexadezimaler Farbcode von Typ CARDINAL, der sich zusammensetzt aus:

- Rotkomponente der Farbe (0H–0FH, also bis 15)
- Grünkomponente der Farbe (0H–0FH, also 0 bis 15)
- Blaukomponente der Farbe (0H–0FH, also 0 bis 15)

Die Farbe Schwarz hat somit den Code 0000H, ein reines Blau hingegen 000FH.

Somit sieht unsere Copperliste folgendermaßen aus:

```
1. CWAIT(Copperliste, 128, Ø);   (* Warte auf Bildschirmmitte *)
2. CMOVE(Copperliste, custom.Color[Ø], ØFØØH); (* Farbwechsel *)
3. CEND (Copperliste, Ø, Ø);                          (* ENDE *)
```

Jetzt ist es nicht schwer, ein Programm, das unser Copperprogramm installiert und ausführt, zu schreiben. Hier ist es:

```
(* Demoprogramm zum Generieren einer einfachen Copperliste *)

MODULE Copper;

FROM SYSTEM     IMPORT ADR,INLINE;
FROM Intuition IMPORT RethinkDisplay,NewScreen,ScreenPtr,OpenScreen,CloseScreen;
FROM Graphics   IMPORT UCopListPtr,ViewPortPtr;
FROM GfxMacros IMPORT CMOVE,CWAIT,CEND;
FROM Hardware   IMPORT custom;
FROM Dos        IMPORT Delay;
FROM Exec       IMPORT AllocMem,MemReqs,MemReqSet;
```

```
VAR MyScreen      : POINTER TO NewScreen;
    MyScreenPtr   : ScreenPtr;
    MyViewPort    : ViewPortPtr;
    MyCopperlist  : UCopListPtr;

(* Screendaten *)

PROCEDURE screenData; (* $E- NewScreen *)
  BEGIN
    INLINE(Ø,Ø,64Ø,256,2,ØØØ1H,8ØØØH,ØØØFH,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø)
  END screenData;

(* Hauptprogramm *)

BEGIN
  MyScreen:=ADR(screenData);
  MyScreen^.defaultTitle:=ADR("Eine einfache Copperliste");
  MyScreenPtr:=OpenScreen(MyScreen^);

  MyViewPort:=ADR(MyScreenPtr^.viewPort);          (* ViewPort deklarieren*)
    MyCopperlist:=AllocMem(SIZE(MyCopperlist^),MemReqSet{chip,memClear}); (* Spei-
cher für Copperliste anfordern *)
  (* Hier steht die eigentliche Instruktionsliste *)

  CWAIT(MyCopperlist,128,Ø);                      (* auf Bildschirmmitte warten *)
  CMOVE(MyCopperlist,ADR(custom.color[Ø]),ØFØØH); (* Farbe in  Rot ändern *)
  CEND (MyCopperlist,Ø,Ø);                        (* Ende der Copperliste *)

  MyViewPort^.uCopIns:=MyCopperlist;              (* Liste in den Viewport
                                                  einfügen *)

  RethinkDisplay;                                 (* und ausführen! *)

  Delay(3ØØ);                                     (* warten, bis sich der
                                                     Betrachter
                                                     satt gesehen hat! *)
  CloseScreen(MyScreenPtr);                       (* Screen schließen *)

END Copper.                                       (* Das war's!!! *)
```

5.4 Hinweise und Tips

Zunächst ein wichtiger Hinweis zu den Befehlen CMOVE und CWAIT: sie sind zu langsam! Zwischen einem CWAIT und dem darauf folgenden CMOVE vergehen 8 Punkte, d.h. der Rasterstrahl ist dann 8 Punkte weiter, so auch bei zwei aufeinanderfolgenden CMOVEs. Bei zwei aufeinanderfolgenden CWAITs vergehen vier Punkte. Das macht sich allerdings nur bei der Programmierung komplizierter Copperlisten bemerkbar.

Als Tips möchte ich Ihnen noch einige sehenswerte Copperlisten, die Sie im Programm zu 5.3 verwenden können, zeigen:

1. 256 Farben gleichzeitig (Grün- und Blautöne):

```
FOR i:=Ø TO 255 DO
  CWAIT(MyCopperlist,i,Ø);
  CMOVE(MyCopperlist,ADR(custom.Color[Ø]),i);
END;
CEND(MyCopperlist,Ø,Ø);
```

2. 256 Farben gleichzeitig (Gelb- und Rottöne):

```
FOR i:=Ø TO 255 DO
  CWAIT(MyCopperlist,i,Ø);
  CMOVE(MyCopperlist,ADR(custom.Color[Ø]),1ØØØH-i);
END;
CEND(MyCopperlist,Ø,Ø);
```

3. 8 Rottöne:

```
FOR i:=Ø TO 7 DO
  CWAIT(MyCopperlist,i*32,Ø);
  CMOVE(MyCopperlist,ADR(custom.Color[Ø]),ØFBBH-11H*i);
END;
CEND(MyCopperlist,Ø,Ø);
```

Sie können anstatt der Hintergrundfarbe auch die Vordergrundfarbe einsetzen, ganz wie es Ihnen beliebt. Besonders reizvoll ist allerdings die Kombination aus »1« und »2«, wobei man bei »2« als Farbregister die Vordergrundfarbe 1 einsetzt. Versuchen Sie es und zeichnen Sie mittels der Grafikbefehle (z.B. RECTFILL) irgendetwas auf den Bildschirm. Sie werden staunen ...

Kapitel 6

Der Blitter

Der Blitter ist, wie der Copper, ein Zusatzchip, der die CPU entlastet. Er verfügt zwar über keine eigene Programmiersprache, mit der er Interrupt-Programme schreiben kann, doch ist er in seinen Operationen unheimlich schnell, ja er ist sogar in der Lage, 125 Kbyte innerhalb einer Sekunde zu kopieren und das Duplikat mit diversen (!) logischen Verknüpfungen zu verändern, wobei wir auch schon bei der Hauptaufgabe des Blitters wären: Speicherbereiche kopieren, löschen und verschieben. Darüber hinaus gibt es noch den sagenumwobenen Zeichenmodus des Blitters für Linien, der allerdings nur über umständliche Assemblerprogrammierung zu erreichen ist. Wir wollen uns hier zuerst mit dem Löschmodus des Blitters beschäftigen.

6.1 Der Löschmodus des Blitters

Um den Blitter zum Löschen von Speicherbereichen zu bewegen, gibt es eigentlich nur eine Möglichkeit, die für uns in Frage kommt: die GRAPHICS-Befehle. Um Speicherbereiche mit dem Blitter zu löschen gibt den Befehl BLTCLEAR:

BltClear (Speicherblock , Bytes , Flags);

Speicherblock	ADDRESSe des zu löschenden Speicherbereichs.
Bytes	LONGCARD-Wert, der die Anzahl der zu löschenden Bytes angibt.
Flags	LONGCARD-Wert, der den Löschmodus angibt. Dieser Parameter sollte normalerweise 1 sein, denn dann wird die tatsächliche Anzahl Bytes, die Sie bei »Bytes« angegeben haben, gelöscht. Eine Alternative dazu wäre der Wert 2: dann geben die unteren 16 Bits (also 0–15) des Parameters »Bytes« die Anzahl der Bytes pro zu löschender Linie an, die oberen 16 aber müssen die Anzahl der zu löschenden Linien enthalten. Der Wert der unteren 16 Bits darf übrigens nicht größer als 128 sein, denn der BLTCLEAR-Befehl kann »nur« Flächen mit einer Größe von 1024*1024 Punkten

löschen (und 128 Byte pro zu löschender Linie * 8 Punkte pro Byte = 1024). Ein Überschreiten dieses Wertes ruft den Guru hervor, was ja nicht Sinn Ihres Programms sein kann.

Als Anwendung für den BLTCLEAR-Befehl kann man z.B. das blitzschnelle Löschen selbst generierter Bitplanes (für Super-Bitmap-Layers oder Dual-Playfields) nennen. Die Größe des zu löschenden Speicherbereichs könnte man mit der Funktion RASSIZE aus GfxMacros errechnen, wenn diese funktionieren würde. Im »ReadMe«-Text auf der Compilerdiskette des M2Amiga-Compilers Version 3.1d fand man unter der Rubrik »Bekannte, aber noch nicht behobene Fehler« schreckensstarr die Bemerkung, daß RASSIZE einen zu großen Wert zurückliefern würde. Bei den Anwendungen, für die wir RASSIZE bisher gebraucht haben, machte sich das nicht bemerkbar, doch bei den Blitter-Befehlen meditiert der AMIGA eine Runde und fragt sich, warum man wohl zuviel Speicher löschen wollte. Um den Programmierer auf dieses Problem aufmerksam zu machen, schickt er ihm eine Guru-Meditation (wie freundlich!).

Aber auch dieses Manko ist nicht so schwerwiegend, daß wir es nicht beheben könnten. Man muß sich dazu nur klarmachen, was der RASSIZE-Befehl eigentlich tut, nämlich die Anzahl der Bytes innerhalb eines Rasters (das kann eine Bitplane, ein temporäres Raster für AREAs, oder auch vieles anderes sein) berechnen, und das kann man eigentlich auch ohne ihn, denn wir wissen, daß 8 Punkte in einer Zeile ein Byte sind. Also müssen wir die Anzahl der Gesamtpunkte durch 8 dividieren und den Quotienten mit der Höhe in Bildpunkten multiplizieren, da das Raster auch eine Höhe haben muß:

*Groesse := (Breite DIV 8) * Höhe*

Nun sind Sie in der Lage, blitzschnell einen Bildschirm zu löschen.

6.2 Kopieren mit dem Blitter

Jetzt kommen wir zu einer weiteren famosen Eigenschaft des Blitters: das Kopieren von Datenblöcken. Die GRAPHICS-Library unterstützt hier Bitmaps als Datenblöcke, obwohl man natürlich auch alle anderen Datenblöcke mit dem Blitter kopieren kann, doch das ist eigentlich nur über Maschinensprache möglich. Wie dem auch sei, für das Kopieren von Bitmaps (oder Ausschnitten aus Bitmaps) gibt es den Befehl BLTBITMAP:

ok := BltBitMap (QuellMap , x1 , y1 , ZielMap , x2 , y2 , Breite ,
Höhe , Minterm , Maske , TempA);

QuellMap BitMapPtr auf die Bitmap, von der kopiert werden soll.

x1	INTEGER-x-Koordinate der linken, oberen Ecke des zu kopierenden Ausschnittes.
y1	INTEGER-y-Koordinate der linken, oberen Ecke des zu kopierenden Ausschnittes.
ZielMap	BitMapPtr auf die Bitmap, in die kopiert werden soll.
x2	INTEGER-x-Koordinate der linken, oberen Ecke des Zielbereichs.
y2	INTEGER-y-Koordinate der linken, oberen Ecke des Zielbereichs.
Breite	INTEGER-Wert, der die Breite des zu kopierenden Ausschnittes angibt (muß zwischen 1 und 976 Punkten liegen).
Höhe	INTEGER-Wert, der die Höhe des zu kopierenden Ausschnittes angibt (muß zwischen 1 und 1023 Punkten liegen).
Minterm	BYTE-Wert, der die Kopierbedingung(en), die wir gleich noch ausführlich besprechen werden, angibt.
Maske	Kopiermaske, in der steht, welche Bitplanes vom Kopieren beeinflußt werden. Jedes Bit repräsentiert eine Bitplane, ist es gesetzt, wird die Plane beeinflußt, ist es gelöscht, wird sie in Ruhe gelassen. Üblicherweise wird hier der Wert 255 (=11111111) angegeben, weil alle Bitplanes vom Kopieren betroffen sein sollen. Dieser Parameter dient dazu, Farbveränderungen vorzubeugen, die eintreten, weil der Blitter Bitplane für Bitplane kopiert. Siehe auch SETWRMASK im GRAPHICS-Kapitel.
TempA	Dieser Parameter zeigt auf einen von Ihnen allozierten Speicherbereich, der benutzt wird, um den Quellausschnitt zu sichern, sollten Sie innerhalb einer Bitmap kopieren und Quell- sowie Zielbitmap sich überschneiden. Der Speicherbereich muß so groß wie der überlappende Bereich sein, aber es ist wohl das Beste, Sie allozieren gleich einen zusätzlichen Speicher mit der Größe des zu kopierenden Ausschnittes. Kopieren Sie nicht innerhalb einer Bitmap oder sind Sie sicher, daß keine Überschneidungen eintreten, geben Sie hier einfach NIL an.
ok	LONGCARD-Wert, der angibt, ob das »Blitten« geklappt hat (0) oder ob irgendetwas schief gelaufen ist (1).

Aber jetzt zu den schon angesprochenen Minterms. Sie enthalten die Kopierbedingungen als BYTE-Wert, die angeben, was mit der Kopie geschehen soll. Es gibt vier Grundminterms, die aber beliebig miteinander über OR, das logische ODER, verknüpft werden können (Kenntnisse in der Boolschen Algebra sind hier ganz nützlich):

```
A=Quelle; B=Ziel

Minterm           Wert
A AND B           Ø8ØH (#128)
A AND NOT B       Ø4ØH (#64)
NOT A AND B       Ø2ØH (#32)
NOT A AND NOT B   Ø1ØH (#16)
```

Daraus kann man dann noch einige nützliche Verknüpfungen ableiten, z.B.:

(NOT A AND NOT B) OR (A AND NOT B) = NOT B entspricht 040H AND 010H = 50H (die Kopie wird invertiert (jede Bitplane für sich, was zu mehr oder minder starken Farbverzerrungen kommt!))

(A AND B) entspricht 080H (Quellpunkte werden nur gesetzt, wenn die Zielpunkte schon gesetzt waren.)

(A AND B) OR (A AND NOT B) = A entspricht 080H AND 040H = 0C0H (es wird eine 1:1 Kopie erzeugt. Dies ist wohl die häufigste Anwendung.)

Eine Unterart des BLTBITMAP-Befehls ist CLIPBLIT, der nicht gleich die Bitmap-Daten kopiert, sondern die entsprechenden RastPorts heranzieht. CLIPBLIT ist vor allem in Intuition-Umgebung (Intuition-Windows, -Screens) zu gebrauchen, da zu großer Kopierbereich nicht gleich mit einem Guru quittiert wird. Was nicht ins Fenster (oder auf den Screen) paßt, wird abgeschnitten (»geclippt«). Hier die Syntax von CLIPBLIT:

ClipBlit (QuellPort , x1 , y1 , ZielPort , x2 , y2 , Breite,
Höhe , Minterm);

QuellPort	RastPortPtr auf den RastPort, von dem kopiert werden soll.
x1	INTEGER-Wert, der die x-Koordinate der linken, oberen Ecke des zu kopierenden Ausschnittes angibt.
y1	INTEGER-Wert, der die y-Koordinate der linken, oberen Ecke des zu kopierenden Ausschnittes angibt.
ZielPort	RastPortPtr auf den RastPort, in den kopiert werden soll.
x2	INTEGER-Breite des Kopierbereichs.
y2	INTEGER-Höhe des Kopierbereichs.
Minterm	BYTE-Wert, der den Minterm beim Kopieren angibt.

(Eine Maske muß nicht angegeben werden, da die RastPorts, die mit SETWRMASK gesetzt werden kann, benutzt wird. Der temporäre Sicherheitsspeicher wird von Intuition zur Verfügung gestellt.)

Was kann man nun Schönes mit diesen Befehlen anstellen? Gagprogramme, beispielsweise. Kopieren Sie den Workbench-Screen 1:1 in einen eigenen und treiben Sie Ihre Späße damit. Sind Sie von einigermaßen humorvoller Natur, dürfte es Ihnen nicht schwer fallen, Verzerrungen des »Pseudo-Workbench-Screens« zu programmieren. Sie können ihn wie schmelzendes Eis zerfließen lassen, langsam auflösen, die Windows der Reihe nach ausblenden (»out-fading«) und vieles andere mehr. Zwei Beispielprogramme sollen Ihnen demonstrieren, wie:

```
MODULE BlitterDemo1;

FROM SYSTEM        IMPORT ADR;
FROM Intuition     IMPORT OpenWindow,CloseWindow,NewWindow,WindowPtr,WindowFlags,
                          WindowFlagSet,IDCMPFlagSet,IDCMPFlags,IntuiMessage,
                          customScreen,OpenScreen,CloseScreen,NewScreen,ScreenPtr,
                          ScreenFlags,ScreenFlagSet;
FROM Graphics      IMPORT ViewModes,ViewModeSet,BltBitMap,WritePixel,SetAPen;
FROM Exec          IMPORT GetMsg,ReplyMsg,WaitPort;
FROM RandomNumber IMPORT RND;

VAR MeinWindow : WindowPtr;
    MeinScreen : ScreenPtr;
    WindowDaten: NewWindow;
    ScreenDaten: NewScreen;
    class      : IDCMPFlagSet;
    IntuiMsg   : POINTER TO IntuiMessage;
    ok         : LONGCARD;

BEGIN
  WITH ScreenDaten DO
    leftEdge:=0; topEdge:=0; width:=640; height:=256; depth:=4; detailPen:=0;
    blockPen:=1; viewModes:=ViewModeSet{hires}; type:=customScreen; font:=NIL;
    defaultTitle:=NIL; gadgets:=NIL; customBitMap:=NIL;
  END;

  WITH WindowDaten DO
    leftEdge:=0;topEdge:=10; width:=100; height:=10;
    detailPen:=0; blockPen:=1; idcmpFlags:=IDCMPFlagSet{closeWindow};
    flags:=WindowFlagSet{windowDrag,windowDepth,windowSizing,activate,
           windowClose};
    firstGadget:=NIL; checkMark:=NIL; title:=ADR("!$%&/+#&");
    bitMap:=NIL; minWidth:=0; maxWidth:=0; minHeight:=0; maxHeight:=0;
    type:=ScreenFlagSet{wbenchScreen};
  END;
```

```
  MeinWindow:=OpenWindow(WindowDaten);
  ScreenDaten.defaultTitle:=MeinWindow^.wScreen^.defaultTitle;
  MeinScreen:=OpenScreen(ScreenDaten);

  ok:=BltBitMap(ADR(MeinWindow^.wScreen^.bitMap),Ø,Ø,ADR(MeinScreen^.bitMap),
          Ø,Ø,64Ø,256,ØCØH,ØFFH,NIL);

  SetAPen(ADR(MeinScreen^.rastPort),Ø);

  LOOP
    ok:=WritePixel(ADR(MeinScreen^.rastPort),RND(64Ø),RND(256));
    IntuiMsg:=GetMsg(MeinWindow^.userPort);
    WHILE IntuiMsg#NIL DO
      class:=IntuiMsg^.class;
      ReplyMsg(IntuiMsg);
      IF (closeWindow IN class) THEN EXIT; END;
      IntuiMsg:=GetMsg(MeinWindow^.userPort);
    END;
  END;
  CloseWindow(MeinWindow);
  CloseScreen(MeinScreen);
END BlitterDemo1.
```

Und Nummer 2:

```
MODULE BlitterDemo2;

FROM SYSTEM     IMPORT ADR;
FROM Intuition IMPORT OpenWindow,CloseWindow,NewWindow,WindowPtr,WindowFlags,
                      WindowFlagSet,IDCMPFlagSet,IDCMPFlags,IntuiMessage,
                      customScreen,OpenScreen,CloseScreen,NewScreen,ScreenPtr,
                      ScreenFlags,ScreenFlagSet;
FROM Graphics   IMPORT ViewModes,ViewModeSet,BltBitMap,ScrollRaster;
FROM Exec       IMPORT GetMsg,ReplyMsg,WaitPort;

VAR MeinWindow : WindowPtr;
    MeinScreen : ScreenPtr;
    WindowDaten: NewWindow;
    ScreenDaten: NewScreen;
    class      : IDCMPFlagSet;
    IntuiMsg   : POINTER TO IntuiMessage;
    ok         : LONGCARD;
    x,y,i,n    : INTEGER;

BEGIN
  WITH ScreenDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256; depth:=4; detailPen:=Ø;
    blockPen:=1; viewModes:=ViewModeSet{hires}; type:=customScreen; font:=NIL;
    defaultTitle:=NIL; gadgets:=NIL; customBitMap:=NIL;
  END;
```

```
  WITH WindowDaten DO
    leftEdge:=0;topEdge:=10; width:=100; height:=10;
    detailPen:=0; blockPen:=1; idcmpFlags:=IDCMPFlagSet{closeWindow};
    flags:=WindowFlagSet{windowDrag,windowDepth,windowSizing,acti-
vate,          windowClose};
    firstGadget:=NIL; checkMark:=NIL; title:=ADR("!$%&/+#&");
    bitMap:=NIL; minWidth:=0; maxWidth:=0; minHeight:=0; maxHeight:=0;
    type:=ScreenFlagSet{wbenchScreen};
  END;

  MeinWindow:=OpenWindow(WindowDaten);
  ScreenDaten.defaultTitle:=MeinWindow^.wScreen^.defaultTitle;
  MeinScreen:=OpenScreen(ScreenDaten);

  ok:=BltBitMap(ADR(MeinWindow^.wScreen^.bitMap),0,0,ADR(MeinScreen^.bitMap),
              0,0,640,256,0C0H,0FFH,NIL);

  FOR i:=0 TO 45 DO
    x:=16*i;
    ScrollRaster(ADR(MeinScreen^.rastPort),0,i-10,x,0,x+15,255);
  END;
  x:=639; n:=ScreenDaten.height DIV 50;
  FOR i:=0 TO 55 DO
    y:=i*n;
    ScrollRaster(ADR(MeinScreen^.rastPort),25-i,0,0,y,x,y+n-1);
  END;

  LOOP
    WaitPort(MeinWindow^.userPort);
    IntuiMsg:=GetMsg(MeinWindow^.userPort);
    WHILE IntuiMsg#NIL DO
      class:=IntuiMsg^.class;
      ReplyMsg(IntuiMsg);
      IF (closeWindow IN class) THEN EXIT; END;
      IntuiMsg:=GetMsg(MeinWindow^.userPort);
    END;
  END;
  CloseWindow(MeinWindow);
  CloseScreen(MeinScreen);
END BlitterDemo2.
```

6.3 Zeichnen mit dem Blitter

Mit dem Befehl BLTTEMPLATE ist es möglich, Elemente aus einem bestimmten Datenblock in die Bitmap zu schreiben. Ein Beispiel für so einen Datenblock sind die

Zeichensätze (Fonts) des Amiga, die Bit an Bit (nicht Byte an Byte!) im Speicher stehen. So ist es ganz leicht möglich, Proportionalschriften, bei denen das »i« schmaler ist als das »m«, zu generieren. Auf anderen Computern, wie dem C64, IBM-PC z.B., kann man nicht ohne weiteres einen proportionalen Schriftsatz generieren, nur um Ihnen die Exklusivität des BLTTEMPLATE-Befehls zu demonstrieren. Es ist aber auch möglich, einen Spezialzeichensatz zu definieren und damit die einzelnen Levels eines Spiels aufzubauen – diese Methode ist unheimlich speichersparend. Die Syntax von BLTTEMPLATE lautet:

BltTemplate (Datenblock , Position , Größe , MeinRastPort ,
x , y , Breite , Höhe);

Datenblock	ADDRESSe des Datenblocks, aus dem die Daten geholt werden sollen (CHIP-Memory!).
Position	INTEGER-Wert, der die Position des gewünschten Zeichens im Datenblock angibt.
Größe	Größe des Datenblocks in Byte.
MeinRastPort	RastPortPtr auf den RastPort, der das Zeichen darstellen soll.
x	INTEGER-Wert, der die x-Koordinate der Position des Zeichens auf dem Bildschirm angibt.
y	INTEGER-Wert, der die y-Koordinate der Position des Zeichens auf dem Bildschirm angibt.
Breite	INTEGER-Zahl, die die Breite des Zeichens in Bildpunkten (also Bits) enthalten muß.
Höhe	INTEGER-Zahl, die die Höhe des Zeichens in Bildpunkten enthalten muß.

Um nun ein entsprechendes Programm zu schreiben, muß man erst den Datenblock definieren. Das geschieht im Prinzip wie bei den Images, zeilenweise (die verschiedenen Zeichen können, wie schon erwähnt, sofort aufeinanderfolgen):

```
        1.Word                           ...
        1.Byte            2.Byte         ...
        8 4 2 1  8 4 2 1  8 4 2 1  8 4 2 1 ...
1.Zeile 1 Ø Ø 1  Ø Ø 1 Ø  Ø 1 Ø Ø  1 Ø Ø 1 ...
2.Zeile Ø 1 1 Ø  1 1 Ø 1  1 Ø 1 1  Ø 1 1 Ø ...
  ...                                    ...

INLINE (Ø9249H, ...);
```

Dann ist es ratsam, einen speziellen ARRAY mit der Position und Breite jedes Elements im Datenblock (nicht auf dem Bildschirm!) zu initialisieren, um diese dann bequem dem BLTTEMPLATE zu übergeben.

Hier ist ein Beispielprogramm, das mit Hilfe des Blitters zeichnet:

```
MODULE TemplateDemo;

FROM SYSTEM     IMPORT ADR,INLINE;
FROM Intuition IMPORT NewWindow,WindowPtr,OpenWindow,WindowFlagSet,ScreenFlags,
                      CloseWindow,IDCMPFlags,IDCMPFlagSet,WindowFlags,
                      ScreenFlagSet;
FROM Graphics   IMPORT BltTemplate,RastPortPtr,Move,SetAPen,SetDrMd,jam1,jam2;
FROM Exec       IMPORT AllocMem,MemReqs,MemReqSet,FreeMem;
FROM Dos        IMPORT Delay;

TYPE CardPtr = POINTER TO CARDINAL;

VAR RP          : RastPortPtr;
    MeinWindow : WindowPtr;
    WindowDaten: NewWindow;
    ChipImage  : CardPtr;
    i          : INTEGER;
    Pos,Breite : ARRAY[Ø..7] OF INTEGER;

PROCEDURE ChipCopy(source: CardPtr; VAR dest: CardPtr; size: LONGCARD);
VAR ChipPtr: CardPtr;
    copied : LONGCARD;
  BEGIN
    ChipPtr:=AllocMem(size,MemReqSet{chip});
    dest:=ChipPtr;
    copied:=Ø;
    REPEAT
      ChipPtr^:=source^;
      INC(ChipPtr,2); INC(source,2); INC(copied,2);
    UNTIL copied=size;
  END ChipCopy;

(* Ab hier stehen die Daten für den BLTTEMPLATE-Befehl, die ins CHIP-Memory
   kopiert werden. Sie bilden den Schriftzug "MODULA-2" *)
PROCEDURE Daten; (* $E- *)
  BEGIN
    INLINE(Ø83FFH,ØE86ØH,Ø18Ø3H,ØFCØØH);
    INLINE(ØC586H,Ø186ØH,Ø24ØØH,ØØ4ØØH);
    INLINE(ØA986H,Ø186ØH,Ø42ØØH,ØØ4ØØH);
    INLINE(Ø9186H,Ø186ØH,ØFFFFH,ØFCØØH);
    INLINE(Ø8186H,Ø186ØH,Ø81Ø2H,ØØØØØH);
```

```
    INLINE(Ø8186H,Ø186ØH,Ø81Ø2H,ØØØØØH);
    INLINE(Ø81FFH,ØEFFFH,Ø81Ø3H,ØFCØØH);
  END Daten;

(* Ab hier steht eine Routine, um die 8 Zeichen des Schriftzuges, jedes für
   sich, an der aktuellen Position des Grafikcursors auszugeben.
   Die Texthöhe beträgt 7 Punkte, der Modulo-Wert 8, da der Text 4 Words,
   also 8 Byte breit ist. *)
PROCEDURE DrawIt;
  BEGIN
    FOR i:=Ø TO 7 DO
      BltTemplate(ChipImage,Pos[i],8,RP,RP^.x,RP^.y,Breite[i],7);
      Move(RP,RP^.x+11,RP^.y);
    END;
  END DrawIt;

BEGIN
  ChipImage:=NIL;
  ChipCopy(ADR(Daten),ChipImage,112); (* Daten kopieren *)

  WITH WindowDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256; detailPen:=Ø;
    blockPen:=1; idcmpFlags:=IDCMPFlagSet{}; flags:=WindowFlagSet{borderless};
    firstGadget:=NIL; checkMark:=NIL; title:=NIL; screen:=NIL; bitMap:=NIL;
    type:=ScreenFlagSet{wbenchScreen};
  END;

  MeinWindow:=OpenWindow(WindowDaten);
  RP:=MeinWindow^.rPort;

  (* Im folgenden werden die Arrays für Position und Breite der einzelnen
     Zeichen initialisiert, da es sich um eine Proportionalschrift handelt. *)
  Pos[Ø]:=Ø; Breite[Ø]:=8; Pos[1]:=8; Breite[1]:=6; Pos[2]:=14; Breite[2]:=6;
  Pos[3]:=2Ø; Breite[3]:=6; Pos[4]:=26; Breite[4]:=6; Pos[5]:=32; Breite[5]:=8;
  Pos[6]:=4Ø; Breite[6]:=6; Pos[7]:=46; Breite[7]:=8;

  SetAPen(RP,1); SetDrMd(RP,jam1);      (* JAM1-Zeichnen *)
  Move(RP,2Ø,35); DrawIt;
  Move(RP,2Ø,5Ø); SetDrMd(RP,jam2); DrawIt; (* JAM2-Zeichnen *)
  Move(RP,2Ø,76); SetDrMd(RP,jam1); DrawIt; (* Outline-Zeichnen *)
  Move(RP,2Ø,74); DrawIt;
  Move(RP,21,75); DrawIt;
  Move(RP,19,75); DrawIt;
  Move(RP,2Ø,75); SetAPen(RP,2); DrawIt;

  Delay(5ØØ);

  CloseWindow(MeinWindow);
  FreeMem(ChipImage,112);
END TemplateDemo.
```

6.4 Weitere Blitter-Befehle

Neben den schon genannten, häufig gebrauchten Befehlen gibt es noch einige weitere, weniger gebrauchte, aber dennoch nicht minder interessante.

Fangen wir mit dem BLTBITMAPRASTPORT-Befehl, der eine reine Bitmap in einen RastPort »blittet«, an. Dieser Befehl ist ganz nützlich, wenn man eine zweite Bitmap mit oft benutzten Bilddaten im Speicher hält, ohne sie darzustellen, und dann ab und zu ein Bildelement in einen dargestellten RastPort zu »blitten«:

BltBitMapRastPort (MeineBitmap , x1 , y1 , MeinRastPort , x2 , y2 , Breite , Höhe , Minterm);

MeineBitmap	BitMapPtr auf die Bitmap, von der kopiert werden soll.
x1	INTEGER-Wert, der die x-Koordinate der linken, oberen Ecke des zu kopierenden Ausschnittes angibt.
y1	INTEGER-Wert, der die y-Koordinate der linken, oberen Ecke des zu kopierenden Ausschnittes angibt.
MeinRastPort	RastPortPtr auf den RastPort, in den kopiert werden soll.
x2	INTEGER-Breite des Kopierbereichs.
y2	INTEGER-Höhe des Kopierbereichs.
Minterm	BYTE-Wert, der den Minterm beim Kopieren angibt.

(Eine Maske muß nicht angegeben werden, da die RastPorts, die mit SETWRMASK gesetzt werden kann, benutzt wird.)

Mit dem Befehl BLTPATTERN lassen sich Flächen füllen:

BltPattern (MeinRastPort , Muster , x1 , y1 , x2 , y2 , Anzahl);

MeinRastPort	RastPortPtr auf den RastPort, der für die zu füllende Fläche zuständig ist.
Muster	ADDRESSe des Füllmusters, das ganz normal definiert sein muß (siehe GRAPHICS-Kapitel).
x1	INTEGER-x-Koordinate der linken, oberen Ecke des Füllbereichs.
y1	INTEGER-y-Koordinate der linken, oberen Ecke des Füllbereichs.

x2	INTEGER-x-Koordinate der rechten, unteren Ecke des Füllbereichs.
y2	INTEGER-y-Koordinate der rechten, unteren Ecke des Füllbereichs.
Anzahl	INTEGER-Wert, der die Anzahl Bytes pro Zeile im Füllmuster angibt.

Kapitel 7

Diskfonts

Wie Sie vielleicht wissen, unterstützt das Betriebssystem des Amiga externe, auf Diskette oder Festplatte befindliche Zeichensätze voll. Gerade verschiedene Zeichensätze geben Ihrem Programm einen besonderen Reiz, es gibt kunstvolle, geschwungene, verschnörkelte, fette, dreidimensional wirkende, zierliche oder spezielle Zeichensätze (Fonts). Wichtig ist, daß sich diese Fonts im Pseudo-Device »fonts« befinden, das normalerweise SYS:fonts ist, mit dem CLI-Befehl ASSIGN aber beliebig geändert werden kann. Auch die Befehle zur Unterstützung der Diskfonts sind in einer eigenen Bibliothek angelegt, der »diskfont.library«, die M2Amiga Besitzer wie immer nicht öffnen brauchen. Die nötigen Strukturen sind im Modul DiskFont enthalten.

7.1 Öffnen und Einbinden eines Diskettenzeichensatzes

Um einen Diskettenzeichensatz zu öffnen, ist der Befehl OPENDISKFONT (aus DiskFont) nötig:

MeinFont := OpenDiskFont (MeinAttr);

MeinAttr	TextAttrPtr auf den TextAttr-Record des Fonts (s.u.).
MeinFont	TextFontPtr auf den TextFont-Record des Fonts (s.u.), oder NIL, wenn irgendwas schief gelaufen ist (der Font konnte beispielsweise nicht gefunden werden).

Der TextAttr-Record legt die zum Öffnen nötigen Daten fest:

```
TextAttr=RECORD
  name : ADDRESS      ; Name des Fonts (z.B. "ruby.font")
  ySize: CARDINAL     ; Höhe des Fonts in Bildpunkten (z.B. 12)
  style: FontStyleSet ; Stil des Fonts (unterstrichen, kursiv, ...)
  flags: FontFlagSet  ; Flags des Fonts
END;
```

name Werfen Sie einen Blick ins fonts:-Directory einer Diskette, die noch Fonts enthält, so werden Ihnen die Files, die die Endung ».font« haben auffallen: das sind die eigentlichen Zeichensätze, deren Namen Sie in diesem Feld angeben müssen.

ySize Im fonts:-Directory einer Diskette, die Fonts enthält, sind auch diverse Unterverzeichnisse, die die Namen der im fonts:-Directory befindlichen Zeichensätze tragen. Darin sind einige Dateien enthalten, die nach den Größen, die dieser Font haben kann, benannt sind. Beispiel: im Verzeichnis SYS:fonts/diamond werden Sie, sofern sich der »Diamond« Zeichensatz auf dieser Diskette befindet, die Dateien 12 und 20 finden, was heißt, daß »Diamond« in zwei Größen erhältlich ist: 12 Punkte hoch und 20 Punkte hoch. Die gewünschte Größe müssen Sie in diesem Feld angeben.

style Hier könnten Sie spezielle Darstellungsarten für Ihren Font angeben, würde das hier funktionieren. Da das aber nicht der Fall ist (Systemfehler?), kann der Textstil nur über SETFONTSTYLE geändert werden, aber das ist kein Problem, wie Sie noch sehen werden.

flags Hier können Sie einige fontspezifische Flags angeben, die der Menge FontFlagSet entstammen:

romFont: Der Zeichensatz befindet sich im ROM (im Moment nur bei »topaz« möglich).

diskFont: Der Zeichensatz befindet sich auf Diskette.

revPath: Der Zeichensatz wurde entwickelt, um von rechts nach links zu schreiben (kommt im Hebräischen vor).

tallDot: Der Zeichensatz wurde extra für die 64*256-Punkte-Darstellung entwickelt.

wideDot: Der Zeichensatz wurde extra für die 320*512-Punkte-Darstellung entwickelt.

proportional: Es handelt sich um eine Proportional-Schriftart.

designed: ??

removed: Das System kann nicht auf den Zeichensatz zugreifen, was praktisch bei allen nicht geöffneten externen Fonts der Fall ist.

Nun zum TextFont-Record eines Fonts, auf den OPENDISKFONT einen Zeiger zurückgibt:

```
TextFont=RECORD
  message   : Message       ; Nachrichtenstruktur des Fonts (unwichtig)
  ySize     : CARDINAL      ; Höhe des Fonts
  style     : FontStyleSet  ; Stil-Flags des Fonts
  flags     : FontFlagSet   ; Flags des Fonts
  xSize     : CARDINAL      ; durchschnittliche Zeichenbreite
  baseLine  : CARDINAL      ; Entfernung zur Basiszeile
  boldSmear: CARDINAL       ; Anzahl der Punkte, um die ein Text im Fettschrift-
                              Modus verschmiert gezeichnet werden soll
  accessors: CARDINAL       ; Anzahl der Zugriffe, die diese Struktur
                              aufzuweisen hat
  loChar    : CHAR          ; erste Zeichen im Font
  hiChar    : CHAR          ; letzte Zeichen im Font
  charData  : ADDRESS       ; Adresse der Bilddaten des Fonts (siehe Blitterkapitel)
  modulo    : CARDINAL      ; Anzahl der Bytes, die die Fontdaten belegen
  charLoc   : ADDRESS       ; Adresse des Positionsarrays (siehe Blitterkapitel)
  charSpace: ADDRESS        ; Adresse einer Tabelle für den Zeichenabstand
                              bei Proportionalschriften
  charKern  : ADDRESS       ; Adresse einer Tabelle für die Punkte,

                              die zwischen den Zeichen freigelassen werden sollen
END;
```

Zur Verdeutlichung einiger Felder sollten Sie sich dieses Bild ansehen:

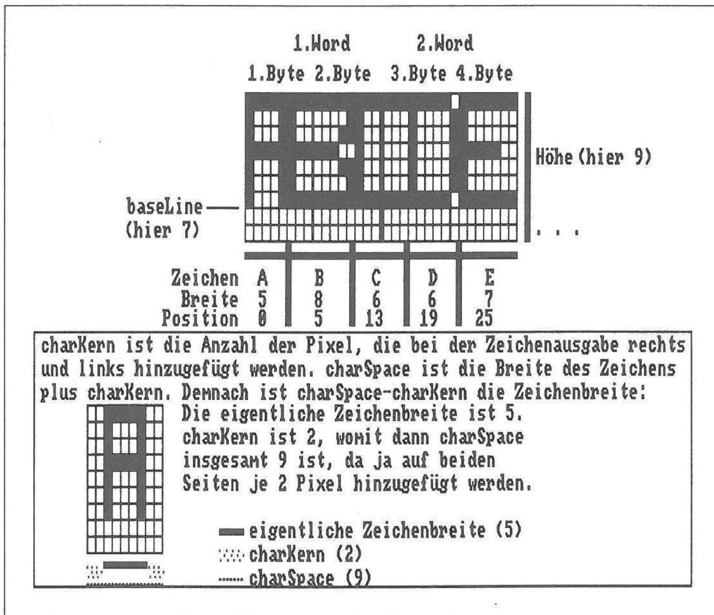


Bild 7.1: Die Fontdaten

Bis jetzt haben wir eigentlich nur das Öffnen von Diskettenfonts besprochen, manchmal ist es aber vonnöten, einen der beiden Systemzeichensätze (»topaz 60« und »topaz 80«, wobei »topaz 60« der größere ist, da man mit ihm nur 60 Zeichen in eine Zeile bekommt, im Gegensatz zu »topaz 80«, der 80 ermöglicht) zu öffnen. Das ist die Aufgabe des OPENFONT-Befehls:

MeinFont := OpenFont (MeinAttr);

MeinAttr	TextAttrPtr auf den TextAttr-Record des zu öffnenden Systemfonts.
MeinFont	TextFontPtr auf den TextFont-Record des soeben geöffneten Fonts. Ist dieser Wert NIL, konnte der Font nicht geöffnet werden (weil er z.B. gar nicht vorhanden war).

Haben Sie Ihren Zeichensatz geöffnet, möchten Sie ihn sicher auch benutzen. Doch wie möchten Sie ihn benutzen? Entweder Sie möchten ihn in einen RastPort einbinden, so daß ab diesem Zeitpunkt jede Textausgabe (durch PRINTITEXT oder TEXT) mit diesem Font geschieht, oder Sie möchten Ihren Font in die Liste des Systems einbinden, damit Sie ihn fortan wie den Systemzeichensatz ansprechen, also in die NewScreen-Struktur einbinden können.

Für die erste Möglichkeit sollten Sie den Befehl SETFONT verwenden:

SetFont (MeinRastPort , MeinFont);

MeinRastPort	RastPortPtr auf den RastPort, in den der Font eingebunden werden soll.
MeinFont	TextFontPtr auf den TextFont-Record des Zeichensatzes, der in den RastPort eingebunden werden soll.

Ist Ihnen die zweite Variation der Fonteinbindung lieber, so ist der ADDFONT-Befehl zu verwenden:

AddFont (MeinFont);

MeinFont	TextFontPtr auf den Zeichensatz, der in die Systemliste eingebunden werden soll.

Nun zur Arbeit mit dem neu geöffneten Font. Es ist irgendwie sinnvoll, die Stilarten eines Fonts aus dem Stehgreif zu ändern, denn man möchte oft nur einzelne Wörter oder Sätze in einer anderen Darstellungsart schreiben. Bevor die Darstellungsart eines

Fonts geändert werden kann, muß erst ermittelt werden, welche Darstellungsarten überhaupt erlaubt sind, denn ist ein Zeichensatz von vornherein unterstrichen, so ist es sinnlos, nochmal eine neue Unterstreichung zu generieren. Die erlaubten Stilarten des aktuellen Zeichensatzes (dieser muß im RastPort eingebunden sein!) kann man mit ASKSOFTSTYLE ermitteln:

Stil := AskSoftStyle (MeinRastPort);

MeinRastPort	RastPortPtr auf den RastPort, dessen aktueller Font auf mögliche Darstellungsmodi geprüft werden soll.
Stil	FontStyleSet-Menge, die die erlaubten Darstellungsmodi enthält. Zulässig sind alle Werte der Menge FontStyleSet: underlined: Der Zeichensatz (bzw. das mit ihm geschriebene) wird unterstrichen dargestellt. bold: Alles, was mit diesem Zeichensatz geschrieben wird, wird fett gedruckt dargestellt. italic: Durch Setzen dieses Flags schalten Sie die Kursiv-(Schräg-)Schrift ein. extended: Der Zeichensatz wird breiter als normal dargestellt. »leere Menge«: der Zeichensatz wird absolut normal dargestellt.

Jetzt erst können mit SETSOFTSTYLE die Darstellungsmodi des aktuellen Fonts geändert werden:

NeuerStil := SetSoftStyle (MeinRastPort , Stil , Erlaubt);

MeinRastPort	RastPortPtr auf den RastPort, dessen aktueller Font in der Darstellungsart verändert werden soll.
Stil	FontStyleSet-Menge, die die neuen Darstellungsmodi enthält.
Erlaubt	FontStyleSet-Menge mit den erlaubten Stilarten. Hier sollte der Rückgabewert von ASKSOFTSTYLE angegeben werden.

Sind Sie einen Font leid oder geht der Speicherplatz zu neige (so ein Font ist recht umfangreich), können Sie ihn mit CLOSEFONT schließen:

CloseFont (MeinFont);

MeinFont	TextFontPtr auf den zu schließenden Font.

Möchten Sie Ihren Font nicht gleich aus dem Speicher, sondern nur aus der Liste des Systems verbannen (er muß natürlich vorher mit ADDFONT eingebunden worden sein), so empfiehlt es sich, REMFONT zu verwenden:

RemFont (MeinFont);

MeinFont	TextFontPtr auf den Font, der aus der Systemliste gestrichen werden soll.

7.2 Spezielle Font-Befehle

Neben den in Kapitel 7.1 schon genannten Hauptfunktionen, die für das Öffnen und Einbinden eines Zeichensatzes unentbehrlich sind, gibt es noch weitere, nicht so wichtige, doch ebenfalls sehr nützliche Befehle. Gerade für die anspruchsvolle Textausgabe ist es oft nötig, zu wissen, wieviele Bildpunkte ein String in einem bestimmten Font einnimmt. Um das festzustellen, ist die Funktion TEXTLENGTH da:

Anzahl := TextLength (MeinRastPort , String , Zeichen);

MeinRastPort	RastPortPtr auf den RastPort, in dem der String, dessen Länge festgestellt werden soll, ausgegeben wird.
String	ADDRESSe des Strings, dessen Anzahl an Bildpunkten ermittelt werden soll.
Zeichen	INTEGER-Wert, der die Anzahl der Zeichen in »String« angibt.
Anzahl	INTEGER-Wert, der die Anzahl der Bildpunkte, die »String« im aktuellen Font geschrieben hat, einnimmt.

Mit dieser Funktion kann ganz einfach zentriert werden, man muß beim MOVE-Befehl bei der x-Koordinate nur folgende Formel anwenden:

x = Breite des Screen DIV 2 – Breite des Strings DIV 2

Schon wird Ihr Text zentriert.
Um die aktuelle Zeile zu löschen, gibt es den Befehl CLEAREOL:

ClearEOL (MeinRastPort);

MeinRastPort	RastPortPtr auf das Window oder den Screen in dem die Zeile, in der der Cursor steht, gelöscht werden soll. Diese Zeile wird mit der Farbe des B-Pens gefüllt.

Möchten Sie beim Löschen nicht bei einer Zeile bleiben, sondern den ganzen Screen löschen, können Sie den Befehl CLEARSCREEN verwenden:

ClearScreen (MeinRastPort);

MeinRastPort	RastPortPtr auf den RastPort, dessen Screen oder Window gelöscht (oder besser mit der Farbe des B-Pens gefüllt) werden soll.

Für manche Anwendungen ist es sinnvoll, den TextAttr-Record des aktuellen Zeichensatzes zu erfahren, wobei wir bei ASKFONT wären:

AskFont (MeinRastPort , MeinAttr);

MeinRastPort	RastPortPtr auf den RastPort, dessen TextAttr-Record geholt werden soll.
MeinAttr	TextAttrPtr auf einen leeren TextAttr-Record, der mit der TextAttr-Struktur des aktuellen Zeichensatzes gefüllt wird.

Möchte man feststellen, auf welche Zeichensätze man überhaupt zugreifen kann, kann man AVAILFONTS (aus DiskFont) verwenden:

ok := AvailFonts (Buffer , Bytes , Modus);

Buffer	ADDRESSe eines genügend großen Speicherbereichs, in dem ein AvailFontHeader (s.u.) eingetragen wird. 1000 Byte dürften genügen.
Bytes	LONGINT-Wert, der die Größe des Speicherbereichs »Buffer« in Bytes angibt.
Modus	BITSET-Wert, der angibt, welche Zeichensätze gesucht werden. Ist Bit 0 gesetzt, so werden alle Zeichensätze im Speicher gesucht, ist Bit 1 gesetzt, wird auf Diskette nach Fonts Ausschau gehalten und sind beide Bits gesetzt, werden Diskette sowie Speicher »durchforstet«.
ok	LONGINT-Rückgabewert, der angibt, ob die Operation erfolgreich verlaufen ist (0). Ist dieser Wert ungleich 0, gibt er die Anzahl der Bytes, um die der Speicherbereich »Buffer« vergrößert werden muß, an.

Kommen wir zu den schon angesprochenen AvailFontHeader, den AVAILFONTS in den angegebenen Speicherbereich schreibt. Dahinter verbirgt sich dieser Record:

```
AvailFontHeader=RECORD
  numEntries: CARDINAL                          ; Anzahl der gefundenen Fonts
  af        : ARRAY[Ø..numEntries-1] OF AvailFont ; AvailFont-Tabelle
END;
```

Die AvailFont-Tabelle besteht aus sovielen Records des Typs AvailFont wie gefundenen Zeichensätzen:

```
AvailFont=RECORD
  type: BITSET   ; Typ des Fonts: ROM-Font (Bit Ø) oder Disk-Font (Bit 1)
  attr: TextAttr ; TextAttr-Record des Fonts
END;
```

Zum Abschluß dieses Kapitels soll ein Beispielprogramm folgen: MODULE FontDemo.

```
(* Achtung! Die Fonts "RUBY" und "GARNET" müssen zum fehlerfreien Betrieb
   dieses Programms auf der Startdiskette befindlich sein, oder es muß mit
   ASSIGN eine andere Font-Diskette definiert werden. *)

FROM SYSTEM     IMPORT ADR,INLINE;
FROM Intuition  IMPORT NewWindow,WindowPtr,OpenWindow,WindowFlagSet,ScreenFlags,
                CloseWindow,IDCMPFlags,IDCMPFlagSet,WindowFlags, ScreenFlagSet;
FROM Graphics   IMPORT RastPortPtr,TextFontPtr,TextAttr,FontStyles,FontFlags,
                FontStyleSet,FontFlagSet,SetAPen,Move,Text,TextLength,
                SetFont,CloseFont,SetDrMd,jam1,AskSoftStyle,SetSoftStyle;
FROM DiskFont   IMPORT OpenDiskFont;
FROM Dos        IMPORT Delay;
FROM Strings    IMPORT Length;

VAR RP          : RastPortPtr;
    MeinWindow  : WindowPtr;
    WindowDaten: NewWindow;
    MeinFont    : TextFontPtr;
    MeinAttr    : TextAttr;
    Style,Mask  : FontStyleSet;
    MeinString  : ARRAY[Ø..5Ø] OF CHAR;

BEGIN
  WITH WindowDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=64Ø; height:=256; detailPen:=Ø;
    blockPen:=1; idcmpFlags:=IDCMPFlagSet{}; flags:=WindowFlagSet{borderless};
    firstGadget:=NIL; checkMark:=NIL; title:=NIL; screen:=NIL; bitMap:=NIL;
```

```
  type:=ScreenFlagSet{wbenchScreen};
END;

MeinWindow:=OpenWindow(WindowDaten);
RP:=MeinWindow^.rPort;

SetDrMd(RP,jam1);  (* JAM1 wegen Schatten *)

WITH MeinAttr DO   (* Attribut-Record initialisieren *)
  name:=ADR("ruby.font"); ySize:=15; style:=FontStyleSet{};
  flags:=FontFlagSet{};
END;

MeinFont:=OpenDiskFont(ADR(MeinAttr)); (* Font öffnen *)
IF MeinFont#NIL THEN    (* ok? *)
  SetFont(RP,MeinFont);   (* ja — setzen *)
  Mask:=AskSoftStyle(RP);   (* Darstellungsmodi holen *)
  Style:=SetSoftStyle(RP,Mask,FontStyleSet{italic}); (* Stil setzen *)
  MeinString:="Amiga is it!";
  SetAPen(RP,2);     (* Schatten zeichnen *)

  Move(RP,320-TextLength(RP,ADR(MeinString),Length(MeinString))/2,120);
  Text(RP,ADR(MeinString),Length(MeinString));
  SetAPen(RP,1);     (* Text zeichnen *)

  Move(RP,(320-TextLength(RP,ADR(MeinString),Length(MeinString))/2)+2,122);
  Text(RP,ADR(MeinString),Length(MeinString));
  CloseFont(MeinFont);    (* Font schließen *)
END;

WITH MeinAttr DO   (* Attribut-Record initialisieren *)
  name:=ADR("garnet.font"); ySize:=16; style:=FontStyleSet{};
  flags:=FontFlagSet{};
END;

MeinFont:=OpenDiskFont(ADR(MeinAttr)); (* Font öffnen *)
IF MeinFont#NIL THEN    (* ok? *)
  SetFont(RP,MeinFont);   (* ja — setzen *)
  Mask:=AskSoftStyle(RP);   (* Darstellungsmodi holen *)
  Style:=SetSoftStyle(RP,Mask,FontStyleSet{underlined,bold});(* Stil setzen *)
  MeinString:="MODULA-2";
  SetAPen(RP,2);     (* Schatten zeichnen *)

  Move(RP,320-TextLength(RP,ADR(MeinString),Length(MeinString))/2,140);
  Text(RP,ADR(MeinString),Length(MeinString));
  SetAPen(RP,1);     (* Text zeichnen *)
```

```
    Move(RP,(320-TextLength(RP,ADR(MeinString),Length(MeinString))/2)+2,142);
    Text(RP,ADR(MeinString),Length(MeinString));
    CloseFont(MeinFont);    (* Font schließen *)
  END;

  Delay(500);

  CloseWindow(MeinWindow);
END FontDemo.
```

Kapitel 8

EXEC

EXEC ist der Teil des Betriebssystems, der für die Steuerung des Multitaskings und allem, was dazugehört (Speicherverwaltung, Verständigung zwischen einzelnen Tasks, Steuerung der Schnittstellen zur Hardware, Bibliotheken, ...) zuständig ist und auf den alle anderen aufbauen. Da das ein sehr komplexes und umfassendes Thema ist, ist auch die EXEC-Programmierung nicht leicht. Die Schnittstelle zum Programmierer ist ist EXEC-Bibliothek, bzw. das EXEC-Modul in Modula-2. Die EXEC-Bibliothek ist aber ständig geöffnet (sie enthält ja auch die Befehle zum Öffnen und Schließen der anderen Libraries), ihre Basisadresse ist »4«. Die Struktur der ExecBase enthält einige sehr interessante Felder:

thisTask (TaskPtr)	Zeiger auf den Task-Record des gerade laufenden Programms memList-taskWait: einige Systemlisten, die die betreffenden Strukturen enthalten

Für weitere Informationen sehen Sie bitte in der Beschreibung des Definitionsmoduls Exec Ihres Compilerhandbuches nach.

8.1 Speicherverwaltung

Eine weitere Hauptaufgabe von EXEC ist, neben der Multitasking-Steuerung, die Speicherverwaltung, was bei mehreren scheinbar gleichzeitig ablaufenden Programmen ganz schön problematisch werden kann. Hier sollen nun die Befehle, die mit Speicherverwaltung zu tun haben, erklärt werden.

Fangen wir mit der ALLOCMEM-Funktion, die wir schon unzählige Male im Verlauf dieses Buches benutzt haben, an. Wie Sie sicher wissen, stellt ALLOCMEM eine angegebene Anzahl von Bytes, die zudem eine bestimmte Eigenschaft haben müssen, im Speicher zur Verfügung:

Speicher := AllocMem (Größe , Bedingungen);

Größe — LONGINT-Wert, der die Größe des zu allozierenden Speichers in Bytes angibt.

Bedingungen — Bedingungen, die der zu allozierende Speicherbereich erfüllen muß. Die Werte der Menge MemReqSet sind zulässig:

public: Der Speicherbereich darf nicht verschoben werden und muß mehreren Tasks gleichzeitig zur Verfügung stehen. Diese Option ist zwar noch nicht implementiert, sollte aber aus Kompatibilitätsgründen für bestimmte Bereiche gewählt werden.

chip: Der Speicherbereich muß in den untersten 512 Kbyte liegen. Nur auf Chip-Speicher können z.B. der Copper und das GRAPHICS-System zugreifen. Momentan sind maximal nur 512 Kbyte CHIP-Speicher verfügbar (trotz aller Speichererweiterungen), doch Gerüchten nach soll bald eine Version des Customchips AGNUS auf den Markt kommen, der dann 1 Megabyte CHIP-Speicher verwalten kann. Vermutlich wird er erst zusammen mit der Version 1.4 des Betriebssystems ausgeliefert.

fast: Der zu allozierende Speicherbereich muß unbedingt im Fast-Memory liegen, auf das GRAPHICS, der Copper und einige andere Systemeinheiten leider nicht zugreifen können. Jedes Byte Speicher, das durch eine Speichererweiterung (A 501 von Commodore, z.B.) gewonnen wird, gehört zum Fast-Memory. Ein Standard-512 Kbyte-Amiga 500 besitzt kein Fast-Memory.

mr3–mr15: für zukünftige Erweiterungen reserviert.

memClear: der Speicherbereich wird bei der Allozierung gelöscht.

largest: es wird der größte zur Verfügung stehende Speicherblock reserviert.

Speicher — ADDRESSe des allozierten Speichers oder NIL, wenn ein Fehler auftrat.

Ein mit ALLOCMEM allozierter Speicher sollte unbedingt mit FREEMEM wieder freigegeben werden, da er sonst bis zu einem Reset nicht mehr benutzbar ist, was sich gerade bei großen Speicherblöcken störend auswirkt:

FreeMem (Speicher , Größe);

Speicher	ADDRESS-Wert, der die Adresse des freizugebenden Speichers angibt.
Größe	LONGINT-Wert, der die Größe des Speichers in Bytes angibt.

Möchten Sie einen Speicherbereich an einer bestimmten Adresse allozieren, müssen Sie den Befehl ALLOCABS verwenden:

Speicher := AllocAbs (Größe , Adresse);

Größe	LONGINT-Wert, der die Größe des Speichers in Bytes angibt.
Adresse	ADDRESS-Wert, der die Adresse des zu allozierenden Speicherbereichs angibt.
Speicher	ADDRESS-Wert, der die Adresse des tatsächlich allozierten Speichers angibt oder NIL bei einem Fehler. Es kann sein, daß diese Adresse von der gewünschten abweicht. Das ist dadurch zu erklären, daß an der gewünschten Adresse nicht genügend Speicher frei ist.

Es wird bei diesem Befehl übrigens immer erst versucht, an Fast-Memory heranzukommen. Auch mit ALLOCABS allozierter Speicherbereich sollte mit FREEMEM wieder freigegeben werden.

Und zum Schluß noch eine spezielle Funktion, die die Größe des größtmöglichen Speicherbereichs mit den angegebenen Bedingungen zurückliefert: AVAILMEM.

Größe := AvailMem (Bedingungen);

Bedingungen	MemReqSet-Wert, der die Bedingungen, die der Speicherbereich haben muß. Siehe dazu auch ALLOCMEM.
Größe	LONGINT-Wert, der die Größe des größten Speicherbereichs mit den Bedingungen »Bedingungen« angibt.

8.2 Nodes

Nodes (Knoten) kommen im EXEC-System sehr häufig vor, um diverse Listen miteinander zu verketten, denn in EXEC ist so ziemlich alles in einer Liste angelegt, was bei einem multitaskingfähigen Betriebssystem ja nicht verwunderlich ist. Der Node-Record hat folgenden Aufbau:

```
Node=RECORD
  succ: NodePtr  ; Zeiger auf nachfolgenden Node
  pred: NodePtr  ; Zeiger auf vorherigen Node
  type: NodeType ; Art des Nodes
  pri : UByte    ; Priorität
  name: ADDRESS  ; Name des Nodes
END;
```

type In diesem Feld wird die Art des Nodes festgelegt. Die Werte des Aufzählungstyps NodeType sind zulässig:

unknown: Unbekannter Node-Typ.

task: Es handelt sich um einen Task-Node.

interrupt: Der Node ist ein Interrupt-Node.

device: Es handelt sich um einen Device-Node.

msgPort: Nachrichtenport-Node.

message: Nachrichten-Node.

freeMsg: freeMsg-Node.

replyMsg: replyMsg-Node.

resource: resource-Node.

library: library-(Bibliotheks-)Node.

memory: Speicher-Node.

softInt: softInt-Node.

font: Zeichensatz-Node.

process: Prozeß-Node.

semaphore: semaphore-Node.

signalSem: signalSem-Node.

Wenn Sie einige Node-Arten noch nicht richtig einordnen können, machen Sie sich keine Gedanken, wir werden das alles noch klären.

pri In diesem Feld wird die Priorität des Nodes angegeben, ist aber nur in einigen Fällen, wie Task- oder Interrupt-Nodes sinnvoll.

name Hier steht die Adresse eines Strings mit dem Namen des Nodes. Ich würde hier einen eindeutigen Namen wählen, z.B. »Drucker-Task«.

Soviel fürs Erste zu den Nodes, die uns durch das ganze EXEC Kapitel begleiten werden. Wie sie initialisiert werden, lernen Sie dann in den einzelnen Unterkapiteln kennen.

8.3 Listen

Listen sind eigentlich nur verkettete Node-Records eines bestimmten Typs, denen ein Listenkopf vorangeht, der List genannt wird:

```
List=RECORD
  head    : NodePtr  ; Zeiger auf ersten Node der Liste
  tail    : NodePtr  ; ist immer NIL
  tailPred: NodePtr  ; Zeiger auf letzten (gültigen) Node der Liste
  type    : NodeType ; Typ der Nodes in dieser Liste
  pad     : BYTE     ; unbenutzt (für spätere Erweiterungen reserviert?)
END;
```

Achtung: Der letzte Node einer Liste muß im Feld succ immer einen Zeiger auf den Listenkopf enthalten, die Nodes untereinander müssen über succ und pred miteinander verbunden sein.

Für die Verwaltung der Listen hat EXEC einige ganz brauchbare Befehle parat, angefangen mit INSERT, der einen Node in eine Liste einfügt:

Insert (MeineListe , MeinNode , Vorgänger);

MeineListe	ListPtr auf die Liste, in die der Node eingefügt werden soll.
MeinNode	NodePtr auf den Node, der in die Liste eingefügt werden soll.
Vorgänger	NodePtr auf den vorhergehenden Nodes des einzufügenden. Wird hier NIL oder der Wert aus dem Feld head des List-Records angegeben, wird der Node »MeinNode« an die erste Stelle der Liste gesetzt (nach dem Listenkopf natürlich). Geben Sie hier den Wert aus dem Feld tailPred des List-Records an, wird der Node »MeinNode« an die letzte Stelle der Liste eingefügt.

Um einen Node gleich an die erste Stelle in der Liste zu setzen, ist der Befehl ADDHEAD da:

AddHead (MeineListe , MeinNode);

MeineList	ListPtr auf die Liste, an dessen erste Stelle der Node »MeinNode« eingefügt werden soll.
MeinNode	NodePtr auf den einzufügenden Node.

Auch ans Ende einer Liste können Sie einen Node einfügen, am besten mit ADDTAIL:

AddTail (MeineListe , MeinNode);

MeinListe	ListPtr auf die Liste, an dessen Ende der Node »MeinNode« eingefügt werden soll.
MeinNode	NodePtr auf den Node, der ans Ende der Liste »MeineListe« gehängt werden soll.

Das Gegenteil zu INSERT ist REMOVE, der einen Node aus einer Liste entfernt:

Remove (MeinNode);

MeinNode	NodePtr auf den zu entfernenden Node.

Auch ADDHEAD hat ein Gegenstück, REMHEAD, der den ersten Node aus einer Liste entfernt:

MeinNode := RemHead (MeineListe);

MeineListe	ListPtr auf die Liste, dessen erster Eintrag entfernt werden soll.
MeinNode	NodePtr auf den entfernten Node.

Um den letzten Node einer Liste zu entfernen, ist es am einfachsten, REMTAIL zu benutzen:

MeinNode := RemTail (MeineListe);

MeineListe	ListPtr auf die Liste, dessen letzter Eintrag entfernt werden soll.
MeinNode	NodePtr auf den Node, der entfernt wurde.

Möchten Sie einen Node nach seiner Priorität geordnet in eine Liste einfügen, sollten Sie ENQUEURE verwenden. Hat schon ein Node die Priorität des neu einzufügenden, wird der »Neuling« hinter den schon eingebundenen Node geschoben:

Enqueure (MeineListe , MeinNode);

MeineListe	ListPtr auf die Liste, in der »MeinNode« eingefügt werden soll.
MeinNode	NodePtr auf den Node, der nach seiner Priorität geordnet in die Liste eingefügt werden soll.

Ist es Ihr Wunsch, einen bestimmten Eintrag aus einer Liste herauszusuchen, können Sie sich des FINDNAME-Befehls bedienen, der einen Eintrag mit angegebenem Namen sucht:

MeinNode := FindName (MeineListe , Name);

MeineListe	ADDRESSe der Liste, die nach einem bestimmten Node durchsucht werden soll.
Name	ADDRESSe des Strings, der den Namen des zu suchenden Nodes enthält.
MeinNode	NodePtr auf einen eventuell gefundenen Node mit dem Namen »Name«, oder NIL, wenn kein Eintrag des angegebenen Namens gefunden wurde.

8.4 Es geht rund – Tasks

Im folgenden wollen wir uns mit der Programmierung und Handhabung des Multitasking befassen. Ein Task (dt. Aufgabe) ist eigentlich nichts weiter als ein Programm, das neben den anderen läuft und vom Prozessor eine bestimmte Zeit, in der es arbeiten darf, zugeschrieben bekommt. Starten Sie ein Programm durch Anklicken von der Workbench oder mit dem CLI-Befehl RUN, bekommt es einen eigenen Task zugeschrieben. Auch die Programmierung eigener Coroutinen mit dem Coroutines-Modul läuft über das Multitasking, das letzendlich EXEC in der Hand hat.

Jeder Task besitzt einen eigenen Record in der Systemliste:

```
Task=RECORD
  node      : Node         ; Node des Tasks (Typ TASKNODE)
  flags     : TaskFlagSet ; Flags des Tasks
  state     : TaskState    ; Zustand des Tasks
  idNestCnt : BYTE         ; Zähler für DISABLE (kommt später)
  tdNestCnt : BYTE         ; Zähler für FORBID (kommt noch)
  sigAlloc  : LONGSET      ; belegte Signale
```

```
  sigWait    : LONGSET     ; Signale, auf die gewartet wird
  sigRecvd   : LONGSET     ; empfangene Signale
  sigExcept  : LONGSET     ; Signale, die Exceptions auslösen
  trapAlloc  : BITSET      ; zugeordnete Traps
  trapAble   : BITSET      ; freigegebene Traps
  exceptData: ADDRESS      ; Datenbereich für Exceptions
  exceptCode: PROC         ; Programm für Exceptions
  trapData   : ADDRESS     ; Datenbereich für Traps
  trapCode   : PROC        ; Programm für Exceptions
  spReg      : ADDRESS     ; Stack des Tasks
  spLower    : ADDRESS     ; Stackuntergrenze
  spUpper    : ADDRESS     ; Stackobergrenze
  switch     : PROC        ; Abgabeprogramm
  launch     : PROC        ; Übernahmeprogramm
  memEntry   : List        ; Speicherliste, in der der Node des Tasks abgelegt ist
  userData   : ADDRESS     ; Benutzerroutine
END;
```

flags Dieses Feld bestimmt, was der Task macht, wenn er an der Reihe ist, ausgeführt zu werden, oder ob er einen Trapfehler ausgelöst hat (»Software failure ...«). Die Werte in der Menge TaskFlagSet sind zulässig:

procTime: Wird in zukünftiger Betriebssystemversion benutzt.

tf1–tf3: Reserviert

stackChk: Wird in zukünftiger Betriebssystemversion benutzt.

exception: Der Task steckt in einem Ausnahmezustand, in den er beispielsweise durch ein bestimmtes Signal versetzt wurde.

switch: Ist dieses Flag gesetzt, wird, wenn sich die CPU einen anderen Task vornimmt und dieser wartet, bis er wieder dran kommt, die im Feld switch bezeichnete Routine angesprungen.

launch: Bei gesetztem LAUNCH-Flag ruft der Task, ist er mit der Abarbeitung an der Reihe, die im Feld launch bezeichnete Routine auf.

state Hier wird der aktuelle Zustand des Tasks angegeben. Die Werte der Aufzählung TaskState sind möglich:

inval: Der Task ist ungültig, da (wahrscheinlich) abgestürzt.

added: Der betreffende Task wird gerade zu den anderen hinzugefügt und in die Systemliste eingereiht.

run: Der Task wird gerade abgearbeitet. Möchte man von seinem eigenen Programm sämtliche RUN-Tasks abfragen, findet man logischerweise stets seinen eigenen.

ready: Der entsprechende Task ist klar zur Abarbeitung, die CPU beschäftigt sich aber noch mit einem anderen.

wait: Der Task wartet auf ein bestimmtes Signal, z.B. ein IDCMP. Durch den Befehl WAITPORT, der schon bei der IDCMP-Abfrage angesprochen wurde, wird der Task, in dem dieser Befehl vorkommt, automatisch in den WAIT-Status versetzt.

except: Der Task hängt in einer Exception.

removed: Der Task wird gerade aus der Systemliste entfernt.

sigAlloc Für jedes Signal (IDCMP-Abfragen, z.B.), auf das gewartet wird, ist ein Bit frei. Die oberen 16 Bits des LONGSET Wertes (16–31) stehen Ihnen für eigene Anwendungen zur Verfügung, während die unteren (0–15) durch das System belegt werden. In diesem Feld sind die einzelnen Bits der Signale verzeichnet, die der Task für sich beansprucht. Es ist aber möglich, das alle oberen 16 Bits schon durch andere Tasks belegt sind.

sigWait Hier stehen die Bits der Signale, auf die der Task augenblicklich wartet (im WAIT-Status ist).

sigRecvd Sollte der betreffende Task ein Signal, auf das er gewartet hat, empfangen, wird das entsprechende Bit hier gesetzt.

sigExcept In diesem Feld sind die Bits der Signale, die einen Ausnahmezustand auslösen, verzeichnet.

trapAlloc Hier stehen die Bits der Traps, die verboten wurden (also keinen Guru, bzw. »Software failure« auslösen).

trapAble Dieses Feld enthält die Bits der noch erlaubten Traps, die einen »Software failure« auslösen.

exceptData Hier steht die Adresse der Daten, die für eine Taskausnahme wichtig sind.

exceptCode In diesem Feld steht ein Zeiger auf eine Routine, die sehr wichtig für die Ausnahmebehandlung bei Tasks ist.

trapData Dieses Feld enthält die Adresse der Trapnummer (siehe auch Anhang A) eines eventuell ausgelösten Trapfehlers.

trapCode Hier steht ein Zeiger auf eine eventuell vorhandene Routine, die zur Trapvermeidung dient, ein eigener Trap-Handler praktisch.

spReg Wird einem anderen Task von der CPU Rechenzeit zugewiesen, speichert EXEC hier die Adresse des Taskstacks. Bei der Erzeugung eines eigenen Tasks müssen Sie die Adresse des Stacks unbedingt hier eintragen. Achtung: der Taskstack muß größer als 71 Byte sein, Sie sollten ihn aber vor allen Dingen bei größeren Programmen auf 10000 Byte setzen, um irgendwelchen scheinbar urplötzlichen und unbegründeten Abstürzen vorzubeugen.

spLower In diesem Feld steht die Adresse der Untergrenze des Taskstacks.

spUpper Hier steht die Adresse der Obergrenze des Taskstacks.

switch Ist bei einem Task das Flag SWITCH gesetzt, wird, wenn die CPU einen anderen Task übernimmt, das hier angegebene Programm angesprungen.

launch Ist bei einem Task das Flag LAUNCH gesetzt, wird, wenn die CPU diesen Task übernimmt, die hier angegebene Routine angesprungen.

memEntry In diesem Feld steht die Liste der vom Task belegten Speicherbereiche.

userData Dieses Feld ist dem Benutzer vorbehalten, der dort beispielsweise einen Zeiger auf eine eigene Routine unterbringen kann.

Hier noch einige generelle Befehle zum Multitasking: Um das Multitasking komplett abzuschalten, was Sie vor der Analyse eines bestehendes Tasks tun sollten, um auch korrekte Daten zu erhalten, ist der Befehl FORBID da:

Forbid;

Es kursiert das Gerücht, dieser Befehl brächte einen gewissen Gewinn an Geschwindigkeit, da ja nur noch das abschaltende Programm läuft, doch dieser ist so minimal, daß es sich nur bei extrem zeitaufwendigen Programmen, wo an jeder Mikrosekunde gespart werden soll, lohnt, das äußerst praktische Multitasking aus Geschwindigkeitsgründen abzuschalten.

Das konkrete Gegenstück zu FORBID ist PERMIT, der das Multitasking wieder erlaubt:

Permit;

Möchte man zusätzlich zum Multitasking auch die Systeminterrupts abschalten, genügt der Befehl DISABLE, der aber niemals über längere Zeit eingeschaltet bleiben darf!

Disable;

Um DISABLE auszuschalten, was man nach so kurzer Zeit wie möglich wieder tun soll, ist der Befehl ENABLE da:

Enable;

Jetzt haben auch die Felder idNestCnt und tdNestCnt eine Bedeutung: In ihnen steht die Anzahl der verwendeten FORBID-/PERMIT-Aufrufe (tdNestCnt) und die der DISABLE-/ENABLE-Aufrufe (idNestCnt).

Nun kommt ein Beispielprogramm, um den Umgang mit Tasks zu verdeutlichen:

```
MODULE TaskDemo;

FROM SYSTEM IMPORT ADR;
FROM Exec   IMPORT execBase,Task,Forbid,Permit;
FROM InOut  IMPORT WriteString,WriteCard,WriteHex,WriteLn;

VAR DieserTask: Task;
    Name      : POINTER TO ARRAY[Ø..15] OF CHAR;

BEGIN
  Forbid;       (* Multitasking verbieten *)
  DieserTask:=execBase^.thisTask^; (* DIESEN Task holen *)
  Permit;       (* Multitasking erlauben *)
  Name:=DieserTask.node.name;     (* Namen holen *)

  WriteLn;
  WriteString("Adresse   Pri  Name"); WriteLn;
  WriteString("------------------"); WriteLn;
  WriteHex(ADR(DieserTask),1);           (* Adresse schreiben *)
  WriteCard(DieserTask.node.pri,7);          (* Priorität schreiben *)
  WriteString("  ");
  WriteString(Name^);            (* Namen schreiben *)
  WriteLn;
END TaskDemo.
```

8.4.1 Eigene Tasks

Man kann natürlich auch eigene Tasks erzeugen. Da mir aber keine vernünftige Anwendung, die man nicht auch mit dem Coroutines-Modul einfacher, komfortabler und vor allem sicherer hätte gestalten können, einfiel, spreche ich dieses Thema nur ganz kurz an.

Um einen neuen Task zu starten, muß genügend Stackspeicher (PUBLIC-Memory!) alloziert und eine leere Task-Struktur bereitgestellt werden. in dieser Struktur brauchen nur die Felder spLower, spUpper, spReg und selbstverständlich node ausgefüllt zu

werden. Das Feld node muß hierbei einen Node-Record enthalten, der einen Node vom Typ TASK und eventuell einen treffenden Namen enthält. Im Feld spLower müssen Sie dann die Adresse des von Ihnen allozierten Stackspeichers (bei sehr, sehr einfachen Task Routinen am besten 1000 Kbyte) eintragen, ins Feld spUpper kommt die Stackobergrenze, die sich wie folgt berechnen läßt:

Obergrenze := Stackadresse + Stackgröße;

In das Record-Feld spReg muß als Anfangsadresse die Stackobergrenze (!) eingetragen werden, da ja die Stackoperationen von oben nach unten verlaufen (LIFO- oder UPN-Prinzip).

Um dann den eigentlichen Task einzubinden, genügt es, den ADDTASK-Befehl aufzurufen:

AddTask (MeinTask , initialPC , finalPC);

MeinTask	TaskPtr auf den Task-Record des Tasks, den Sie initialisieren möchten.
initialPC	ADDRESSe der Routine, die Sie als eigenen Task einbinden möchten.
finalPC	ADDRESSe einer eventuellen Rücksprungroutine, die aufgerufen wird, wenn der Task beendet ist. Wird hier NIL angegeben, verwendet EXEC seine eigene Rücksprungroutine, die den Task aus der internen Liste entfernt und den Speicher, den der Task belegte, freigibt.

Nun zum Entfernen von Tasks: hierzu dient der Befehl REMTASK:

RemTask (MeinTask);

MeinTask	TaskPtr auf den Task-Record des zu entfernenden Tasks.

Man bedenke, was sich damit anstellen läßt: Bekommt man irgendwie die Adresse der Task-Struktur des zu entfernenden Tasks heraus, kann man so noch offene Windows oder Screens, deren Programme vor einigen Zeitaltern abgestürzt sind einfach schließen. Oder man kann Programme, die aufwendige Berechnungen durchführen und aus sich heraus nicht abgebrochen werden können, elegant aus dem Speicher entfernen. Dank des Listensystems ist es auch kein Problem, die Adressen der einzelnen Tasks herauszufinden. Allerdings dürfen Sie auf keinen Fall in der Systemliste rumschmieren und schon gar nicht ohne DISABLE und ENABLE (es gibt doch tatsächlich Leute, die

sich darüber aufregen). Um einen Task sauber aus dem System zu patchen, ist wesentlich mehr Aufwand nötig, als einfach in der internen Liste nach ihm zu suchen. Wie man Tasks »sauber« entfernt, ist nicht Sache dieses Buches, dazu müßten Sie sich ziemlich tief in die DOS- und CLI-Strukturen hinabbegeben, CLI-Segmente löschen (mit UNLOADSEG), dem Console-Device-Task eine Abschiedsmeldung zuschicken und so weiter.

8.4.2 Weitere Task-Befehle

Neue Tasks lassen sich überaus einfach mit dem CREATETASK-Befehl aus ExecSupport generieren, was eine echte Alternative zu ADDTASK ist:

MeinTask := CreateTask (Name , Priorität , Programm , Stack);

Name	ADDRESSe des Strings mit dem Tasknamen.
Priorität	Byte-Wert, der die Priorität des Tasks enthält.
Programm	ADDRESSe der Routine des Tasks
Stack	LONGINT-Wert, der die Größe des Task-Stacks in Bytes enthält.
MeinTask	TaskPtr auf den Task-Record des neuen Tasks.

Der entsprechende Befehl zum Entfernen eines mit CREATETASK eingeführten Tasks lautet DELETETASK und ist ebenfalls in ExecSupport befindlich:

DeleteTask (MeinTask);

MeinTask	TaskPtr auf den zu entfernenden Task.

Es gibt noch zwei weitere, nicht besprochene Task-Befehle, nämlich FINDTASK und SETTASKPRI. FINDTASK sucht in der internen Liste nach einem Task angegebenen Namens:

Task := FindTask (Name);

Name	ADDRESSe des Strings mit dem Namen des Tasks. Gibt man hier NIL an, so bekommt man die Struktur des eigenen Tasks zurückgeliefert.
TaskTask	Ptr auf den gesuchten Task, falls einer gefunden wurde. Ist das nicht der Fall, enthält dieser Wert NIL.

Der zweite Befehl, SETTASKPRI, ändert die Priorität des angegebenen Tasks, ähnlich dem CLI-Befehl CHANGETASKPRI:

AltePriorität := SetTaskPri (MeinTask , NeuePriorität);

MeinTask	TaskPtr auf den Task, dessen Priorität verändert werden soll.
NeuePriorität	UByte-Wert, der die neue Priorität des Tasks »MeinTask« angibt.
AltePriorität	UByte-Wert, der die alte Priorität des Tasks »MeinTask« enthält.

8.4.3 Das Nachrichtensystem der Tasks

Damit sich die Tasks untereinander und den Benutzer verständigen können, wurde ein hervorragend funktionierendes Nachrichtensystem entwickelt.

8.4.3.1 Signale

Ein wesentlicher Bestandteil des Kommunikationssystems der Tasks sind die Signale. Wie schon erwähnt, stehen insgesamt 32 Bits für die Signalbenutzung zur Verfügung, wovon allerdings die unteren 16 (0–15) für Systemsignale reserviert sind. Die anderen 16 kann der Programmierer für sein Programm nutzen. So können Sie sich für Ihren Task ein Signal bereitstellen lassen und darauf warten, daß irgendein anderer Task dieses setzt. Der Vorteil dieser Methode ist, daß ein Task, der auf Signale wartet, auf die Warteliste gesetzt wird (er bekommt das State-Flag WAIT) und keine wertvolle Rechenzeit verbraucht. Würden Sie Nachrichten von anderen Tasks in Form einer Schleife abfragen (busy wait – aktives warten), würde Ihr Task auf die Liste der laufenden Tasks gesetzt (RUNNING) und somit Rechenzeit verbrauchen, was in einem Multitaskingsystem mit Rücksicht auf andere Tasks vermieden werden sollte.

Um erstmal ein Signal für eigene Anwendungen zu belegen, muß man die Funktion ALLOCSIGNAL verwenden:

ok := AllocSignal (MeinSignal);

MeinSignal	LONGINT-Wert, der die Nummer des zu besetzenden Signals enthält oder –1, falls man das nächstfreie Signal belegen möchte.
ok	LONGINT-Rückgabewert, der entweder die Nummer des belegten Signals enthält oder –1 ist, wenn kein Signal mehr frei war.

Das Ganze hätte natürlich wenig Sinn, wenn man nicht auf Signale warten könnte. Man kann es mit WAIT. WAIT setzt den betreffenden Task auf die Warteliste und verharrt,

bis ein anderer Task eins der angegebenen Signale gesetzt hat. Dann wird das Programm weiter abgearbeitet. Die Syntax von WAIT:

Signale := Wait (Maske);

Maske	LONGSET-Wert, der angibt, auf welche Signale, die durch die einzelnen Bits des SET-Wertes repräsentiert werden, gewartet werden soll.
Signale	LONGSET-Wert, der, wenn ein anderer Task ein Signal, auf das gewartet wurde, setzte, das auftretende Signal (bzw. die auftretenden Signale) enthält. Mittels einer Abfrage in der Form

```
IF Signal1 IN Signale THEN ... (* Signal1 wurde gesetzt *)
```

kann man bequem herausfinden, welches Signal gesetzt wurde und dementsprechende Reaktionen hervorrufen.

Übrigens wird das Taskswitching, sofern es durch FORBID oder DISABLE ausgeschaltet wurde von WAIT wieder aktiviert, da sich das System sonst in einer Endlosschleife aufhängen würde. Setzen Sie das in eigenen Programmen aber bitte nie voraus, und gewöhnen Sie sich bitte einen sorgfältigen und sauberen Programmierstil an.

Möchte man ein alloziertes Signal wieder freigeben und für die anderen Tasks zugänglich machen, ist der Befehl FREESIGNAL zu benutzen:

FreeSignal (MeinSignal);

MeinSignal	LONGINT-Wert, der die Nummer des Signals, das freigegeben werden soll, enthält.

Es gibt auch die Möglichkeit, Signale zu verändern. SETSIGNAL macht's möglich:

AlteSignale := SetSignal (NeueSignale , Signale);

NeueSignale	LONGSET-Wert, der den neuen Zustand der Signale angibt. Ein Signalbit kann jedoch nur verändert werden, wenn dessen Bit in »Signale« gesetzt ist.
Signale	LONGSET-Wert, der angibt, welche Signale verändert werden dürfen.
AlteSignale	LONGSET-Wert, der die alten Signale enthält.

Mit dem Befehl

Signale := SetSignal(LONGSET{},LONGSET{});

ist es möglich, die bestehenden Signale in die Variable »Signale« zu bekommen. Dann kann ganz einfach abgefragt werden, ob sie mit denen, auf die gewartet werden soll, übereinstimmen. Diese Technik haben wir schon bei der einfachen IDCMP-Abfrage im Intuition-Kapitel kennengelernt. Durch die IDCMP-Flags werden bestimmte Signalbits im Nachrichtenport des betreffenden Fensters gesetzt und können mit den augenblicklichen Signalen, die durch die oben aufgeführte Variante des SETSIGNAL-Kommandos ermittelt werden, verglichen werden.

Um für einen anderen Task ein oder mehrere Signale zu setzen, ist es ratsam, den Befehl SIGNAL zu verwenden:

Signal (Task , Signale);

Task	TaskPtr auf den Task, für den die Signale gesetzt werden sollen.
Signale	LONGSET-Wert, der die Signale, die gesetzt werden sollen, enthält.

Hat der angesprochene Task mit WAIT auf die Signale gewartet, kehrt er aus dem WAIT-Modus in den READY- oder RUNNING-State zurück und wird weiter abgearbeitet.

8.4.3.2 Das Nachrichtensystem

Nicht nur mit Bits wie die Signale, sondern mit beliebigen Daten arbeitet das Nachrichtensystem von EXEC. Es übernimmt das komplette Senden und Empfangen von Nachrichten und auch die Verwaltung eines Nachrichtenspeichers, in dem überfällige Nachrichten gespeichert werden, falls der empfangende Task sie nicht schnell genug beantwortet.

Als Grundlage für die Nachrichtenverarbeitung existiert ähnlich dem ViewPort oder RastPort ein Nachrichtenport, MsgPort genannt:

```
MsgPort=RECORD
  node       : Node              ; Node des Ports
  CASE flags: MsgPortAction OF ; Flag des Ports
  |signal    :                   ; Port für einen Task
    sigBit  : UByte             ; Signalbit
    sigTask : TaskPtr           ; Task für das Signal
  |softint   :                   ; Port für einen Interrupt
    padØ    : BYTE              ; reserviert
    softInt : InterruptPtr      ; Zeiger auf den Interrupt
```

```
  |ignore   :                    ; egal
    pad1    : BYTE               ; reserviert
    pad2    : ADDRESS            ; reserviert
  END;
  msgList   : List               ; Liste mit den Nachrichten
END;
```

flags In diesem Feld steht, welcher Art der Nachrichtenport sein soll. Je nach Eintrag ändert sich der Aufbau des Records. Zulässig für dieses Feld sind alle Werte aus der Aufzählung MsgPortAction:

signal: Wenn das Nachrichtensystem eine Nachricht erhält, werden die Signale aus dem Feld sigBit an den Task, dessen Adresse im Feld sigTask steht, gesendet.

softint: Wenn das Nachrichtensystem eine Nachricht erhält, wird der Interrupt, auf den softInt zeigt, ausgelöst.

ignore: Egal, was für eine Nachricht das System erhält – es passiert überhaupt nichts.

msgList Hier steht die Nachrichtenliste.

Bevor wir jetzt voll in das Nachrichtenwesen einsteigen können, müssen wir erst einen eigenen Nachrichtenport schaffen. Dazu bedienen wir uns am besten des Befehls CREATEPORT, der aber nicht zur EXEC-Bibliothek, sondern zur Systemlibrary des Linkers gehört. Beim M2Amiga-System ist er im Modul ExecSupport zu finden:

MeinPort := CreatePort (Name , Priorität);

Name	ADDRESSe des Strings mit dem Portnamen. Dieser Parameter ist nicht unbedingt notwendig, man kann hier auch NIL angeben.
Priorität	Byte-Wert, der die Priorität dieses Ports vor den anderen angibt.

Kommt eine Nachricht an, hat sie stets einen Kopf in der Form, wie sie der Message-Record angibt:

```
Message=RECORD
  node      : Node        ; Node der Message, Typ MESSAGE
  replyPort: MsgPortPtr ; Zeiger auf den antwortenden MsgPort (kommt gleich)
  length    : CARDINAL    ; Länge der Nachricht in Bytes
END;
```

Diesem Record folgt dann die eigentliche Nachricht. Es sind z.B. ganze Texte möglich, sie müssen nur kleiner als 64 Kbyte sein, was aber in den meisten Fällen völlig genügt.

Es ist aber zu beachten, daß eine gesendete Nachricht nicht in den Speicher des empfangenden Tasks kopiert wird, der empfangende Task darf lediglich den entsprechenden Speicherbereich des sendenden Tasks benutzen.

Das Senden einer Nachricht ist Aufgabe des PUTMSG-Befehls:

PutMsg (Port , Nachricht);

Port	MsgPortPtr auf den MsgPort, der die Nachricht empfangen soll.
Nachricht	ADDRESSe der Nachricht, die gesendet werden soll. Sie muß die Form des Message-Records haben und wird nicht in den empfangenden Task kopiert, sondern erlaubt diesem nur, den betreffenden Speicherbereich des sendenden Tasks zu benutzen, um die Nachricht zu lesen.

Wie empfängt man nun eine Nachricht? Das ist fast genauso einfach wie das Senden. Man muß lediglich drei Schritte machen:

1. Warten auf die Nachricht,
2. Empfangen der Nachricht und
3. Beantworten der Nachricht. Und natürlich Auswerten der Nachricht als vierter Schritt, doch das ist ja nachrichtenspezifisch.

Zu 1.: Das Warten auf eine Nachricht erfolgt mit dem WAITPORT-Befehl, der eigentlich eine Steigerung des WAIT-Befehls ist. Dieser Befehl versetzt den Task solange in den WAIT-Modus, bis eine Nachricht den MsgPort erreicht. Danach wird das Programm weiter abgearbeitet.

Zu 2.: Ist eine Nachricht angekommen, kann man diese mit GETMSG holen:

Nachricht := GetMsg (MeinPort);

MeinPort	MsgPortPtr auf den Nachrichtenport, von dem die Nachricht geholt werden soll.
Nachricht	ADDRESSe einer Nachricht im Message-Format oder NIL, wenn überhaupt keine Message im Port stand.

Zu 3.: Bevor man sich ans Auswerten der Nachricht macht, sollte man sie beantworten. Die Nachricht steht ja in einem Speicherbereich des sendenden Tasks und dieser kann den Speicher ja nicht einfach so löschen, ohne zu wissen, daß der Empfänger seine Nachricht zur Kenntnis genommen hat, oder er wartet so-

gar noch auf eine Rückmeldung des Empfängers. So muß man notgedrungen einen Antwort-Nachrichtenport (ReplyPort) einrichten, der aber durchaus mit dem »normalen« MessagePort identisch sein kann. Seine Adresse muß im Feld replyPort des Message-Records stehen. Nun bereitet das Beantworten einer Nachricht keine Probleme mehr, der REPLYMSG-Befehl erledigt es:

ReplyMsg (Nachricht);

Nachricht	ADDRESSe der zu beantwortenden Nachricht. Sie wird an den Antwort-Nachrichtenport, dessen Zeiger im Feld replyPort des Nachrichten-Records steht, geschickt. Die Nachricht kann dabei durchaus verändert oder ganz neu sein. So ist ein Nachrichten-Pendelverkehr zwischen zwei oder mehreren Tasks kein Problem mehr.

Um einen Nachrichtenport anhand seines Namens zu finden, kann man die Funktion FINDPORT benutzen:

MeinPort := FindPort (Name);

Name	ADDRESS-Wert, der die Adresse des Strings mit dem Namen des zu findenden Ports enthält.
MeinPort	MsgPortPtr auf den Nachrichtenport mit dem Namen »Name«, falls einer gefunden wurde, sonst NIL.

Möchten Sie einen Nachrichtenport wieder loswerden und den von ihm belegten Speicher freigeben, ist es am einfachsten, den DELETEPORT-Befehl aus ExecSupport zu benutzen:

DeletePort (MeinPort);

MeinPort	MsgPortPtr auf den zu löschenden Message-Port.

Jetzt soll ein Beispielprogramm kommen, das eine Anwendung von Signalen und Messages zeigt. Es arbeitet mit IDCMPs, die Ihnen noch aus dem Intuition-Kapitel geläufig sein dürften (oder haben Sie es nicht gelesen?) und demonstriert beide Arten, eine IDCMP-Nachricht auszuwerten, nämlich mit Messages und Signalen:

```
MODULE MessageDemo;

FROM SYSTEM     IMPORT LONGSET;
FROM Exec       IMPORT GetMsg,ReplyMsg,WaitPort,SetSignal;
```

```
FROM Intuition IMPORT WindowPtr,IntuiMessagePtr,ModifyIDCMP,IDCMPFlags,
                      IDCMPFlagSet;
FROM Windows    IMPORT OpenWindow,CloseWindow,WinGad,WinGadSet,WriteS;

VAR MeinWindow: WindowPtr;
    Nachricht : IntuiMessagePtr;
    Typ       : IDCMPFlagSet;

BEGIN

  OpenWindow(MeinWindow,Ø,Ø,64Ø,256,"Message-Demo",WinGadSet-{closing});
  WriteS(MeinWindow,"Zum Beenden das Schließsymbol zweimal anklicken!");

  ModifyIDCMP(MeinWindow,IDCMPFlagSet{closeWindow}); (* IDCMP setzen *)

  LOOP(* Schleife *)
    WaitPort(MeinWindow^.userPort);(* auf Message warten *)
    Nachricht:=GetMsg(MeinWindow^.userPort);    (* Message holen *)
    WHILE Nachricht#NIL DO(* Message ungleich NIL ? *)
      Typ:=Nachricht^.class;(* Messagetyp retten *)
      ReplyMsg(Nachricht);(* beantworten *)
      IF (closeWindow IN Typ) THEN EXIT; END;    (* Schleife verlassen *)
      Nachricht:=GetMsg(MeinWindow^.userPort);  (* nächste Message holen *)
    END;
  END;

  LOOP(* noch 'ne Schleife *)
    (* Mit diesem Ausdruck wird getestet, ob sich Signalbit des Window-
       Nachrichtenports gleich dem Signal ist, auf das gewartet werden soll.
       Dieses wird mit SETSIGNAL(LONGSET{},LONGSET{}) ermittelt.
    *)

    IF (MeinWindow^.userPort^.sigBit IN
    SetSignal(LONGSET{},LONGSET{})) THEN
       EXIT;
    END;
  END;

  CloseWindow(MeinWindow);
END MessageDemo.
```

8.5 Die Devices

Neben den »großen« Libraries, die zahlreiche Routinen zur komfortablen Programmierung enthalten, gibt es noch einige weniger umfangreiche »Routinensammlungen«, die Devices. Diese sind die Schnittstellen zu den Gerätetreibern, die schließlich und letztendlich die eigentliche Hardware, wie Diskettenlaufwerk, Drucker, Tastatur usw. steuern. Die einzelnen Devices enthalten, abhängig von dem Gerätetreiber, den sie unterstützen, diverse gerätespezifische Befehle.

Wie die Libraries haben auch die Devices eines gleich – sie werden auf dieselbe Art und Weise geöffnet. Der M2Amiga-Compiler übernimmt übrigens nicht das Öffnen und Schließen der benutzten Devices! Ich möchte an dieser Stelle noch sagen, daß nicht alle Devices besprochen werden, da z.B. das Input-Device Kenntnisse in der Assemblerprogrammierung benötigt und das hier ein reines Modula-2 Buch ist.

8.5.1 Öffnen der Devices

Da auch die Devices viel mit Kommunikation zu tun haben (Daten vom Treiber holen, Befehle senden, ...), muß für jedes Devices ein Nachrichtenport initialisiert werden, was man am besten mit CREATEPORT tut. Danach muß ein IORequest-Record initialsiert werden, über den die Befehle und Parameter zum Device gelangen (nicht über einen Funktionsaufruf und Parameterzeile, das ist nur bei den Bibliotheksbefehlen so):

```
IORequest=RECORD
  message: Message   ; Nachricht
  device : DevicePtr ; Zeiger auf den Systemrecord des Devices
  unit   : UnitPtr   ; Zeiger auf evt. Zusatzinformation (z.B.
                       welches von 2 Laufwerken)
  command: CARDINAL  ; Device-Kommando
  flags  : IOFlagSet ; Device-Flags
  error  : Byte      ; Fehlermeldung
END;
```

Um diesen Record brauchen wir uns im Moment nicht weiter zu kümmern, da er von CREATEEXTIO aus ExecSupport initialisiert wird:

MeinIORequest := CreateExtIO (MeinMsgPort , Größe);

MeinMsgPort	MsgPortPtr auf den MsgPort, der für den Device-Verkehr benutzt werden soll.
Größe	INTEGER-Wert, der die Größe des IORequest-Blocks in Bytes enthält.
MeinIORequest	ADDRESSe der fertig initialisierten IORequest-Struktur.

Nun kann das Device mit dem OPENDEVICE-Befehl geöffnet werden:

ok := OpenDevice (Name , Unit , MeinRequest , Flags);

Name	ADDRESS-Wert, der die Adresse des Strings mit dem Namen des Devices enthält.
Unit	LONGINT-Wert, der eine von Device abhängige Nummer enthält, z.B. die Nummer des Laufwerks bei mehreren Floppies.
MeinRequest	ADDRESS-Wert, der die Adresse des mit CREATEEXTIO kreierten IORequest-Blocks enthält.
Flags	LONGSET-Wert, der normalerweise »leere Menge« sein sollte.
ok	LONGINT-Rückgabewert, der entweder 0 (alles klar) oder ungleich 0 (es trat ein Fehler auf) enthält.

Natürlich müssen Devices auch wieder geschlossen werden. Das eigentliche Device schließt man mit CLOSEDEVICE:

CloseDevice (MeinIORequest);

MeinIORequest	ADDRESSe der zuständigen IORequest-Struktur.

Auch die beiden Ports sollten wieder freigegeben werden. Den IORequest-Block löscht man mit DELETEEXTIO aus ExecSupport:

DeleteExtIO (MeinIORequest);

MeinIORequest	ADDRESSe des IORequest-Blocks, der gelöscht werden soll.

Der Nachrichtenport ist mit DELETEPORT, ebenfalls aus ExecSupport, zu löschen:

DeletePort (MeinPort);

MeinPort	MsgPortPtr auf den zu löschenden Nachrichtenport.

8.5.2 Das TrackDisk-Device

Das TrackDisk-Device ist eine Schnittstelle zum Floppy-Treiber, der die Hardware des Diskettenlaufwerks steuert. Das ist zwar über das DOS auch möglich, doch nicht in diesem Maße. Mit dem TrackDisk-Device lassen sich einzelne Blöcke auf der Diskette lesen und schreiben, der Laufwerkstyp kann ermittelt werden, ja sogar komplizierte Kopierprogramme kann man unter Benutzung des TrackDisk-Devices realisieren.

Für den folgenden Verlauf dieses Kapitels ist es wichtig, zu wissen, wie eine 3,5-Zoll-Diskette aufgebaut ist. Grob gesehen ist eine Diskette aus sogenannten Tracks, das sind ineinanderliegende Ringe, aufgebaut. Ein Amiga-Floppylaufwerk nutzt beide Seiten der Diskette, so daß auf jeder Seite Tracks angebracht sind. Liegen sich zwei Tracks gegenüber, werden beide als ein Zylinder bezeichnet. Der Schreib-/Lesekopf, der, wie der Name schon sagt, für das Schreiben und Lesen von Daten zuständig ist, steht immer über einem Zylinder, denn es gibt insgesamt vier solcher Köpfe – zwei für die Diskettenunterseite und zwei für die Oberseite. Die Köpfe werden durch den Schiebemotor (auch Steppermotor genannt) von einem Zylinder zum anderen verschoben, und zwar immer von innen nach außen oder umgekehrt.

Eine normal formatierte Amiga-Diskette besteht grundsätzlich aus 80 frei beschreibbaren Zylindern, die von 0–79 durchnumeriert sind. Zylinder 0 liegt am äußersten Rand der Diskette, Zylinder 79 hingegen am innersten. Da jeder Zylinder zwei Tracks hat (einen oben und einen unten), gibt es folglich 160 Tracks auf einer normalen Disk. Die Tracks mit ungeraden Nummern befinden sich dabei auf der Diskettenunterseite, die mit geraden auf der Oberseite (was anderes bleibt ja kaum übrig). Absolut elitäre Super-Qualitätsdisketten lassen sich auch gelegentlich auf 82 oder 83 Tracks formatieren (was aber oft am Diskettenlaufwerk selber scheitert, da manche im Amiga verwendete Laufwerksmarken nicht mehr als 81 Tracks anfahren können).

Ein Track wird wiederum in 11 Sektoren, die hintereinander auf dem Track liegen, zerlegt, wobei in jeden Sektor 512 Byte passen. Folgende Rechnung beweist die Richtigkeit dieser Aufteilung:

80 Zylinder * 2 Tracks * 11 Sektoren * 512 Byte = 880 Kbyte

Da es aber recht umständlich und vor allen Dingen zeitaufwendig ist, die Position eines Bytes durch Zylinder, Track und Sektor zu erarbeiten, rechnet das TrackDisk-Device intern in Blöcken (Blocks). Es gibt auf einer Diskette 1760 Blöcke (von 0–1759 durchnumeriert). Jeder Block enthält 512 Byte und repräsentiert einen Sektor irgendwo auf der Diskette. Die ersten 11 Blöcke befinden sich auf der Diskettenoberseite in Zylinder 0, Track 0. Die folgenden 11 befinden sich auf der Diskettenunterseite, Zylinder 0, Track 1. Die nächsten sind dann wieder auf der Oberseite, im Zylinder 1, Track 0. Hat man konkrete Zahlen für Zylinder, Sektor und Seite, läßt sich leicht die Nummer des entsprechenden Blocks ermitteln:

Block = 22 * Zylinder + 11 * Seite + Sektor

Die 22 resultiert sich aus 2 Tracks * 11 Sektoren; Seite ist entweder 0 (Oberseite) oder 1 (Unterseite).

8.5.2.1 Programmierung des TrackDisk-Devices

Das TrackDisk-Device wird ganz normal geöffnet, wie Sie es schon kennengelernt haben, nämlich mit dem OPENDEVICE-Befehl (als Namen müssen Sie »trackdisk. device« angeben, als Unit die Nummer des Laufwerks, das Sie ansprechen möchten, 0 für DF0:, 1 für DF1:, ...). Auch die Initialisierung eines Nachrichtenports erfolgt, wie immer, mit CREATEPORT. Nur die Initialisierung eines IORequest-Records ist anders. Da die meisten TrackDisk-Device Befehle recht viele Parameter benötigen, reicht der normale IORequest nicht aus. Ein spezieller IOStdReq muß her:

```
IOStdReq=RECORD
  message: Message    ; Nachricht
  device : DevicePtr ; Zeiger auf das Device
  unit   : UnitPtr   ; Spezialinformation
  command: CARDINAL  ; Device-Befehl
  flags  : IOFlagSet ; Device-Flags
  error  : Byte      ; Device-Fehlermeldung
  actual : LONGCARD  ; befehlsspezifische Information
  length : LONGCARD  ; befehlsspezifische Information
  data   : ADDRESS   ; Datenblock für bestimmte Befehle
  offset : LONGCARD  ; befehlsspezifische Information
END;
```

Die Besprechung der interessanten Felder dieses Records folgt nach und nach jeweils im Kontext.

Auch für das Erstellen so eines Requests gibt es einen Befehl in ExecSupport, nämlich CREATESTDIO:

MeinIOStdReq := CreateStdIO (MeinMsgPort);

MeinMsgPort	MsgPortPtr auf einen zuvor mit CREATEPORT initialisierten Nachrichtenport, der für den Device-Verkehr genutzt wird.
MeinIOStdReq	IOStdReqPtr auf den soeben erstellten IOStdReq und NIL, wenn bei der Erstellung des Requests ein Fehler auftrat.

Der so generierte IOStdReq muß dann beim OPENDEVICE-Befehl anstelle des IO-Requests übergeben werden.

Möchten Sie einen IOStdReq löschen, benutzen Sie am besten den Befehl DELETEST-DREQ aus ExecSupport:

DeleteStdReq (MeinIOStdReq);

MeinIOStdReq	IOStdReqPtr auf den IOStdReq, der gelöscht werden soll.

8.5.2.2 Die TrackDisk-Device-Befehle

Das TrackDisk-Device verfügt über einige spezielle Befehle, deren Codes dem Feld command des IOStdReq-Records übergeben werden müssen. Dann können Sie den Befehl mit DOIO ausführen lassen:

DoIO (MeinRequest);

MeinRequest	ADDRESSe des Requests (IORequest oder IOStdReq).

Es gibt einige Befehle, die in Exec als Konstanten definiert sind und für jedes Device gelten, nur, je nach Device, andere Bedeutungen haben (die in Klammern stehenden Zahlen sind die Codes der Befehle):

```
invalid (#Ø)
reset   (#1)
read    (#2)   aus einem Block lesen
write   (#3)   in einen Block schreiben
update  (#4)   Buffer schreiben
clear   (#5)   Buffer löschen
stop    (#6)
start   (#7)
flush   (#8)
nonstd  (#9)   Basis für devicespezifischen Befehl
```

Die kommentarlosen Befehle haben für das TrackDisk-Device keine Bedeutung. Daneben gibt es noch weitere, devicespezifische Befehle. Die des TrackDisk-Device sind in TrackDisk definiert:

```
motor        (nonstd+Ø)   Steppermotor ein-/ausschalten
seek         (nonstd+1)   Schreib-/Lesekopf positionieren
format       (nonstd+2)   Track formatieren
remove       (nonstd+3)   Interrupt initialisieren
changeNum    (nonstd+4)   Diskettenwechesel holen
changeState  (nonstd+5)   Laufwerksstatus holen
protStatus   (nonstd+6)   Schreibschutzstatus holen
rawRead      (nonstd+7)   Track ohne Decodierung lesen
rawWrite     (nonstd+8)   Track ohne Codierung schreiben
getDriveType (nonstd+9)   Laufwerkstyp holen
getNumTracks (nonstd+1Ø)  Anzahl Tracks holen
addChangeInt (nonstd+11)  Interrupt initialisieren
remChangeInt (nonstd+12)  Interrupt stornieren
lastComm     (nonstd+13)  letztes Kommando holen (noch nicht implementiert)
```

Nun folgt die Besprechung der einzelnen Befehle. Der Befehl READ liest eine Anzahl Bytes von Diskette in einen Datenpuffer im Speicher. Da das TrackDisk-Device zum Datentransfer den Blitter benutzt, muß der Datenpuffer unbedingt im CHIP-Speicher liegen. Folgende Felder des IOStdReq-Records müssen vor dem DOIO-Aufruf belegt sein:

command:read (oder #2)

length	Hier muß die Anzahl der zu lesenden Bytes stehen.
data	In diesem Feld muß die Adresse des Datenspeichers, in den die Bytes eingelesen werden sollen, stehen.
offset	Byte auf Diskette, ab dem zu lesen ist.

Nach dem DOIO-Aufruf sind folgende Felder belegt:

error	Evt. Fehlermeldung, wenn das Lesen nicht geklappt haben sollte.

Zu dem Feld offset bleibt noch zu sagen, daß sich das Byte nach folgender Vorschrift für einen zu lesenden Sektor berechnen läßt:

$$OffsetByte = 512 * (Sektor + 11 * Seite + 44 * Track)$$

Für einen zu lesenden Block:

$$OffsetByte = 512 * Block$$

Nun folgt ein Demoprogramm, das die Bootblöcke (die ersten beiden Blöcke einer Diskette) von der Diskette im Laufwerk DF0: in eine Datei angegebenen Namens schreibt, die dann mit TYPE <name> OPT H ausgegeben werden kann. So können Sie fremde Bootblöcke nach verdächtigen Text (»Something wonderful has happened ...«) untersuchen und feststellen, ob sich ein Virus auf der Diskette befindet. Dann nämlich ist in den Bootblöcken meist ein hämischer Text versteckt. Ein Beispielaufruf für das Demoprogramm wäre:

TrackReadDemo ram:MeinBootBlock

Doch nun zum Programm, für dessen volles Verständnis Sie sich aber den Rest dieses Kapitels ansehen sollten:

```
MODULE TrackReadDemo;

FROM SYSTEM      IMPORT ADR,ADDRESS,LONGSET;
FROM Exec        IMPORT MsgPortPtr,IOStdReqPtr,DoIO,IOStdReq,read,AllocMem,
                        MemReqSet,MemReqs,OpenDevice,CloseDevice,FreeMem;
FROM ExecSupport IMPORT CreatePort,DeletePort,CreateStdIO,DeleteStdIO;
FROM Trackdisk   IMPORT motor;
FROM Dos         IMPORT Open,Close,Write,FileHandlePtr;
FROM Arguments   IMPORT GetArg;

VAR dport     : MsgPortPtr;
    dreq      : IOStdReqPtr;
    ok        : INTEGER;
    buffer    : ADDRESS;
    file      : FileHandlePtr;
    Dateiname: ARRAY[Ø..5Ø] OF CHAR;

(* Prozedur, um den Motor ein- und auszuschalten *)

PROCEDURE Motor(iostdr:IOStdReqPtr;switch:LONGCARD);
  BEGIN
    iostdr^.command:=motor;       (* Kommando: Motorschalten *)
    iostdr^.length:=switch;       (* Flag übergeben *)
    DoIO(iostdr)                  (* ausführen! *)
  END Motor;

(* Prozedur, um den Bootblock zu lesen *)

PROCEDURE
ReadBootBlock(iostdr:IOStdReqPtr;block:LONGCARD;buffer:ADDRESS);
  BEGIN
    iostdr^.command:=read;        (* Kommando: Lesen *)
    iostdr^.length:=1Ø24;         (* Länge: 2 Sektoren *)
    iostdr^.data:=buffer;         (* Zeiger auf Datenspeicher *)
    iostdr^.offset:=512*block;    (* Byte-Offset *)
    DoIO(iostdr)                  (* ausführen! *)
  END ReadBootBlock;

BEGIN
  GetArg(1,Dateiname,ok);   (* Dateinamen holen *)
  buffer:=AllocMem(1Ø24,MemReqSet{chip});(* 512 Byte RAM (wegen Blitter nur
              CHIP-RAM) reservieren *)
  dport:=CreatePort(Ø,Ø);                (* Nachrichtenport des Laufwerks *)
  dreq:=CreateStdIO(dport);              (* Kommunikationsport einrichten *)
  OpenDevice(ADR("trackdisk.device"),Ø,dreq,LONGSET{}); (* TrackDisk öffnen *)
```

```
  Motor(dreq,1);                   (* Motor einschalten (Power!) *)
  ReadBootBlock(dreq,Ø,buffer);    (* Block reinziehen und in der Speicherstelle,
                                      auf die "Buffer" zeigt, speichern *)
  Motor(dreq,Ø);                   (* Motor ausschalten (Power Off!) *)
  CloseDevice(dreq);               (* Trackdisk schließen *)
  DeleteStdIO(dreq);               (* Kommunikationsport auflösen *)
  DeletePort(dport);               (* Nachrichtenport entfernen *)
  file:=Open(ADR(Dateiname),1ØØ6); (* Zielfile öffen (endlich!) *)
  ok:=Write(file,buffer,1Ø24);     (* 1Ø24 coole Bytes ins Zielfile schreiben *)
  Close(file);                     (* Zielfile schließen *)
  FreeMem(buffer,1Ø24);            (* Speicher freigeben *)
END TrackReadDemo.
```

Das Gegenteil zu READ ist WRITE, der den Inhalt eines Datenpuffers (Chip-Memory) auf Diskette schreibt:

command:write (oder #3)

length	Hier muß die Anzahl der zu schreibenden Bytes stehen.
data	In diesem Feld muß die Adresse des Datenspeichers, der auf Diskette geschrieben werden soll, stehen.
offset	Byte auf Diskette, ab dem zu schreiben ist.

Nach dem DOIO-Aufruf sind folgende Felder belegt:

error	Evt. Fehlermeldung, wenn das Schreiben nicht geklappt haben sollte.

Der Wert des Feldes offset läßt sich mit der Methode, die bei dem Befehl READ erläutert wurde, berechnen.

Beim Lesen liest das TrackDisk-Device immer ganze Tracks ein und behält diese in einem speziellen Speicherbereich, dem TrackBuffer, bis ein neuer Track eingelesen wird. Der Befehl UPDATE veranlaßt ein sofortiges Schreiben der Daten des TrackBuffers auf Diskette, falls deren Inhalt geändert wurde:

command: update (oder #4)

Rückgabewerte

error	evt. Fehlermeldung

Das augenblickliche Löschen des TrackBuffers bewirkt der Befehl CLEAR:

command	clear (oder #5)
Rückgabewerte	
error	evt. Fehlermeldung

Kommen wir zu den TrackDisk-spezifischen Befehlen. Vor dem ersten Diskettenzugriff mit dem TrackDisk-Device sollte der Steppermotor eingeschaltet und spätestens am Programmende wieder ausgeschaltet werden. Dafür ist der MOTOR-Befehl da:

command: motor

length	Zustand, in dem der Steppermotor vesetzt werden soll, 0 steht für Ausschalten des Motors, 1 für Einschalten.
Rückgabewerte:	
error	Evt. Fehlermeldung

Um den Schreib-/Lesekopf auf ein bestimmtes Byte zu bewegen ohne irgendeinen Zugriff auf die Diskette vorzunehmen, ist der SEEK-Befehl da:

command: seek

offset	Byte, auf das der Schreib-/Lesekopf bewegt werden soll. Dieses kann auf dieselbe Art und Weise wie beim READ- oder WRITE-Befehl berechnet werden.
Rückgabewerte:	
error	Evt. Fehlermeldung

Ein weiterer TrackDisk-Befehl ist FORMAT, der eine angegebene Anzahl Tracks (und nur soviel) formatiert oder unformatierte Tracks (können auch solche mit Diskettenfehlern sein) mit Bytes füllt:

command: format

length	Gesamtsumme der Bytes der zu formatierenden Tracks
data	Adresse des Datenblocks, der auf in den Track/die Tracks geschrieben werden soll.
offset	Byte-Wert des Starttracks, kann nach der Formel, die Sie bei READ kennengelernt haben, berechnet werden.

Rückgabewerte :

error Evt. Fehlermeldung

Die Gesamtsumme der Bytes der zu formatierenden Tracks, wie sie das Feld length enthalten muß, kann wie folgt berechnet werden:

Gesamtsumme = AnzahlTracks * 5632

(5632 ergibt sich aus 11 Sektoren * 512 Byte.)

Eine ganz spezielle und ausgefallene Funktion hat der Befehl REMOVE. Er installiert einen Interrupt, der immer dann auftritt, wenn eine Diskette aus dem Laufwerk herausgenommen oder hineingelegt wird:

---

command : remove

---

data Adresse der Interrupt-Struktur des betreffenden Interrupts

Rückgabewerte:

error Evt. Fehlermeldung

Der Interrupt kann übrigens durch Angeben von NIL im Feld data und anschließendem Ausführen mit DOIO wieder »ausgeklinkt« werden.

Manchmal kann es ganz interessant sein, die Anzahl der bis jetzt stattgefundenen Diskettenwechsel zu wissen, und genau diese liefert CHANGENUM zurück:

---

command: changeNum

Rückgabewerte :

actual Anzahl der bis jetzt stattgefundenen Diskettenwechsel

error Evt. Fehlermeldung

In vielen Fällen möchte man vom Programm aus kontrollieren, ob überhaupt eine Diskette im Laufwerk liegt. Diese Überprüfung kann man mit CHANGESTATE vornehmen:

command: changeState

Rückgabewerte:

actual	Laufwerksstatus, ist 0, wenn eine Diskette im Laufwerk liegt oder ungleich 0, wenn sich keine im Laufwerk befindet.
error	Evt. Fehlermeldung

Es gibt auch einen Befehl, um den Schreibschutzstatus einer Diskette zu ermitteln. PROTSTATUS heißt er:

command: protStatus

Rückgabewerte:

actual	Schreibschutzstatus der eingelegten Diskette, ist 0, wenn sie beschreibbar oder 255, wenn sie schreibgeschützt ist.
error	Evt. Fehlermeldung

Möchte man einen Track ganz ohne Decodierung durch den Blitter von Diskette einlesen oder auf das Indexloch warten, so ist der Befehl RAWREAD zu verwenden. Mit diesem Befehl ist es möglich, Fremdformate zu lesen (die natürlich entsprechend ausgewertet werden müssen) und beispielsweise ein Kopierprogramm für den Atari ST zu schreiben. Hier ist die Syntax von RAWREAD:

command : rawRead

flags	Hier muß die Konstante indexSync (aus TrackDisk) oder #4 stehen, wenn man auf das Indexloch warten möchte.
length	In diesem Feld muß die Anzahl der zu lesenden Daten (muß unbedingt kleiner als 32 768 sein) stehen.
data	Dieses Feld muß die Adresse des Datenblocks, in den der Track eingelesen werden soll, stehen.
offset	Nummer des zu lesenden Tracks

Rückgabewerte:

error	Evt. Fehlermeldung

Auch zu RAWREAD gibt es ein konkretes Gegenstück, nämlich RAWWRITE. RAWWRITE schreibt einen Datenblock ohne Codierung durch den Blitter auf Diskette oder wartet auf das Indexloch:

command : rawWrite

flags	Hier muß die Konstante indexSync (aus TrackDisk) oder #4 stehen, wenn man auf das Indexloch warten möchte.
length	In diesem Feld muß die Anzahl der zu schreibenden Daten (muß unbedingt kleiner als 32768 sein) stehen.
data	Dieses Feld muß die Adresse des Datenblocks, der auf Diskette geschrieben werden soll, stehen.
offset	Nummer des Tracks, der mit dem Inhalt des Datenpuffers, dessen Adresse im Feld data angegeben werden muß, beschrieben werden soll.

Rückgabewerte:

error	Evt. Fehlermeldung

Den Laufwerkstyp des Diskettenlaufwerks, auf dessen Treiber das TrackDisk-Device zugreift, kann man mit GETDRIVETYPE ermitteln:

command : getDriveType

Rückgabewerte:

actual	Laufwerkstyp, entweder die Konstante drive35 (#1) bei einem 3,5-Zoll-Laufwerk oder drive525 (#2) bei einem 5,25-Zoll-Laufwerk.

Um die Anzahl der Tracks des spezifizierten Laufwerks zu ermitteln, genügt es, den Befehl GETNUMTRACKS zu senden:

command : getNumTracks

Rückgabewerte:

actual	Anzahl der Tracks des spezifizierten Laufwerks.

Kommen wir nun zu den beiden nicht (richtig) funktionierenden TrackDisk-Befehlen, ADDCHANGEINT und REMCHANGEINT. Diese sollten eigentlich mehrere Inter-

rupts, die bei einem Diskettenwechsel angesprungen werden, initialisieren, bzw. diese Interrupts wieder entfernen. Nun faßt das System aber eine im Feld data übergebene Interrupt-Struktur (oder besser einen Zeiger darauf) nicht als solche auf, sondern nimmt den IORequest-Record als solche an, was spätestens beim Schließen des Track-Disk-Devices zum Systemabsturz führt, beim Diskettenwechsel natürlich schon früher.

Und nun zu einem Programm, das die Befehle PROTSTATUS, CHANGESTATE, CHANGENUM, GETDRIVETYPE und GETNUMTRACKS demonstriert, indem es diese auf die Diskette im Laufwerk DF0: anwendet:

```
MODULE TrackDemo;

FROM SYSTEM      IMPORT LONGSET,ADR;
FROM TrackDisk   IMPORT getDriveType,changeNum,changeState,protStatus,
                        getNumTracks;
FROM Exec        IMPORT MsgPortPtr,IOStdReqPtr,OpenDevice,IOStdReq,DoIO,
                        CloseDevice;
FROM ExecSupport IMPORT CreatePort,CreateStdIO,DeleteStdIO,DeletePort;
FROM InOut       IMPORT Read,WriteCard,WriteString,WriteLn;

VAR dport     : MsgPortPtr;
    dreq      : IOStdReqPtr;
    protect,drivetyp,change,tracks: LONGCARD;
    reply     : CHAR;

BEGIN
  dport:=CreatePort(Ø,Ø);     (* Port kreieren *)
  dreq:=CreateStdIO(dport);    (* Request kreieren *)

  OpenDevice(ADR("trackdisk.device"),Ø,dreq,LONGSET{}); (* Device öffnen *)

  dreq^.command:=changeState;    (* prüfen, ob Disk eingelegt *)
  DoIO(dreq);       (* ausführen! *)
  IF (dreq^.actual # Ø) THEN     (* keine Disk? *)
    WriteString("Diskette einlegen und Leertaste+RETURN drücken!");
    REPEAT Read(reply) UNTIL reply=" ";
  END;

  dreq^.command:=protStatus;    (* Schreibschutz ermitteln *)
  DoIO(dreq);      (* ausführen! *)
  protect:=dreq^.actual;    (* Status retten *)
```

```
  dreq^.command:=changeNum;    (* Diskettenwechsel ermitteln *)
  DoIO(dreq);       (* ausführen! *)
  change:=dreq^.actual;      (* Wechsel retten *)

  dreq^.command:=getDriveType;    (* Laufwerkstyp holen *)
  DoIO(dreq);       (* ausführen! *)
  drivetyp:=dreq^.actual;    (* Typ retten *)

  dreq^.command:=getNumTracks;    (* Anzahl Tracks holen *)
  DoIO(dreq);       (* ausführen! *)
  tracks:=dreq^.actual;      (* Tracks retten *)

  (* Nun kommt die Auswertung ... *)
  WriteString("Schreibschutz: ");
  WriteCard(protect,1);
  WriteLn;
  WriteString("Diskwechsel   : ");
  WriteCard(change,1);
  WriteLn;
  WriteString("Drivetyp      : ");
  WriteCard(drivetyp,1);
  WriteLn;
  WriteString("Tracks        : ");
  WriteCard(tracks,1);
  WriteLn;

  (* Arbeit beendet, hinterlasse geordneten Zustand! *)
  CloseDevice(dreq);      (* Device schließen *)
  DeleteStdIO(dreq);      (* Request vernichten *)
  DeletePort(dport);      (* Port vernichten *)
END TrackDemo.
```

8.5.2.3 Die speziellen TrackDisk-Befehle

Nun gibt es aber noch einige ganz spezielle TrackDisk-Befehle, die das Arbeiten mit dem TrackDisk-Device noch komfortabler machen, denn sie prüfen erst, ob der Benutzer seit dem Öffnen des Devices nicht die Diskette gewechselt hat. Somit wird sichergestellt, daß, wenn schon etwas geschrieben wird, es auch auf die richtige Diskette kommt. Die »normalen« TrackDisk-Device-Befehle arbeiten ohne diese Kontrolle, was wertvolle Daten zerstören könnte, wenn ein Track auf die falsche Diskette geschrieben wird. Versucht man, bei Verwendung der Spezialbefehle, auf eine andere Diskette als die, die beim Öffnen des TrackDisk-Devices im Laufwerk war, zu schreiben, erhält man den Fehler Nummer #29 (diskChanged; siehe dazu auch Kapitel 8.5.2.4).

Um die erweiterten Kommandos zu nutzen, muß man anstatt des IOStdReq-Records einen IOExtTD-Record als Request einrichten:

```
IOExtTD=RECORD
  req      : IOStdReq ; IOStdReq-Record
  count    : LONGCARD ; Anzahl der Diskettenwechsel —
                        wichtig für die erweiterten Befehle
  secLabel: LONGCARD ; Kennung
END;
```

Bis auf die beiden Fehler count und secLabel ist dieser Record mit dem IOStdReq identisch. Initialisiert wird er mit dem schon bekannten Befehle CREATEEXTIO aus ExecSupport. Einer IOExtTD Struktur wird man mit dem ebenfalls schon bekannten und besprochenen ExecSupport-Befehl DELETEEXTIO wieder ledig.

Im folgenden sind die erweiterten Befehle aufgelistet, die wie ihre »normalen« Äquivalente arbeiten:

```
extWrite     (write+8ØØØH)     siehe WRITE
extRead      (read+8ØØØH)      siehe READ
extMotor     (motor+8ØØØH)     siehe MOTOR
extSeek      (seek+8ØØØH)      siehe SEEK
extFormat    (format+8ØØØH)    siehe FORMAT
extUpdate    (update+8ØØØH)    siehe UPDATE
extClear     (clear+8ØØØH)     siehe CLEAR
extRawRead   (rawRead+8ØØØH)   siehe RAWREAD
extRawWrite  (rawWrite+8ØØØH)  siehe RAWWRITE
```

8.5.2.4 Die TrackDisk-Fehlermeldungen

Bei fast allen TrackDisk-Befehlen, könnte man als Rückmeldung die Nummer eines Fehlers im Feld error des Request-Records bekommen. Doch was verbirgt sich hinter diesen Nummern, die im TrackDisk-Modul auch als Konstanten mit definiert sind? Diese Auflistung zeigt es:

```
notSpecified   (#2Ø)  Fehler ist nicht spezifizierbar
noSecHdr       (#21)  keinen Sektorkopf gefunden
badSecPreamble (#22)  Fehler in der Sektor-Einleitung
badSecId       (#23)  Fehler in der Sektor-Identifikationsnummer
badHdrSum      (#24)  Prüfsummenfehler im Sektorkopf
badSecSum      (#25)  Prüfsummenfehler im Sektor selbst
tooFewSecs     (#26)  nicht genug Sektoren für einen Track
badSecHdr      (#27)  Sektorkopf zerstört
writeProt      (#28)  Schreiben unmöglich, da Schreibschutz
diskChanged    (#29)  Diskettenwechsel bei Spezialbefehlen (EXTWRITE, z.B.)
seekError      (#3Ø)  Positionierung beim SEEK- oder EXTSEEK-Befehl unmöglich
```

noMem	(#31)	zuwenig freier Speicher
badUnitNum	(#32)	angesprochenes Laufwerk ist defekt oder nicht angeschlossen
badDriveType	(#33)	falscher Laufwerkstyp
driveInUse	(#34)	momentan kein Laufwerkszugriff möglich, da es im Augenblick schon benutzt wird
postReset	(#35)	Zugriffsversuch nach einem Device-Reset

8.5.3 Das Printer-Device

Das Printer-Device dient, wie der Name schon vermuten läßt, zur Ansteuerung des Druckertreibers. Man kann mit diesem Device:

- Druckerkommandos senden
- Texte drucken
- Grafiken drucken

Freilich ohne sich um die verschiedenen Druckertypen zu kümmern, denn für die eigentliche Ausgabe werden die Treiber im Verzeichnis SYS:DEVS/PRINTERS benutzt.

Das Printer-Device wird ganz normal geöffnet, wie das TrackDisk-Device auch. Als Request müssen wir, je nach Druckmodus, stets einen anderen nehmen, doch dazu später.

8.5.3.1 Das Drucken von Texten

Für das Drucken von Texten brauchen wir einen IOStdReq-Record als Request, wie beim TrackDisk-Device. Folgende Werte sind für das Feld command möglich:

write (# Ausgabe von Strings mit Kommandosequenzen

rawWrite (nonstd+0) Ausgabe von Strings ohne Kommandosequenzen

Der Befehl WRITE gibt die im angegebenen Datenpuffer befindlichen Werte, also Zeichen, auf dem Drucker aus. Eventuell im String vorkommende Kommandosequenzen (siehe Kapitel 8.5.3.2) werden berücksichtigt. Hier die Werte für WRITE:

command :	*WRITE (oder #3)*
data	Adresse des auszugebenden Datenblocks
length	Länge des Datenblocks in Bytes
Rückgabewerte:	
error	Evt. Fehlermeldung

Auch RAWWRITE druckt den Inhalt eines Datenspeichers, jedoch nur den puren Text, ohne Kommandosequenzen:

command :	*rawWrite*
data	Adresse des auszugebenden Datenblocks
length	Länge des Datenblocks in Bytes

8.5.3.2 Die Ausgabe von Kommandosequenzen

So ziemlich alle Drucker, die an einen Amiga angeschlossen sind, dürften über diverse Spezialfunktionen, wie Schön-, Fett-, Schmal-, Kursivschrift und viele andere verfügen. Diese werden meist über spezielle Steuer- oder Kommandosequenzen, die oftmals im Anhang des Druckerhandbuchs aufgelistet sind, eingeschaltet. Allgemein durchgesetzt hat sich der sogenannte ESC/P-Standard der Firma Epson. Nun verfügt der Amiga über viele eigene Steuerzeichen, die der Druckertreiber dann in die einzelnen Druckercodes übersetzt, sofern der Drucker das erlaubt. Die Amiga-Steuerzeichen sind im Modul Printer definiert:

Name	Nummer	Kommandosequenz	Wirkung
ris	Ø	ESCc	Druckerreset
rin	1	ESC#1	Druckerinitialisierung
ind	2	ESCD	Zeilenvorschub (Linefeed)
nel	3	ESCE	Return und Zeilenvorschub
ri	4	ESCM	Zeilenvorschub aufwärts
sgrØ	5	ESC[Øm	normaler Schriftstil
sgr3	6	ESC[3m	Kursivschrift (Italic) ein
sgr23	7	ESC[23m	Kursivschrift aus
sgr4	8	ESC[4m	Unterstreichen (Underline) ein
sgr24	9	ESC[24m	Unterstreichen aus
sgr1	1Ø	ESC[1m	Fettschrift (Bold) ein
sgr22	11	ESC[22m	Fettschrift aus
sfc	12	ESC[3Øm-ESC[39m	Vordergrundfarbe (bei Farbdruckern)
sbc	13	ESC[4Øm-ESC[49m	Hintergrundfarbe (bei Farbdruckern)
shorpØ	14	ESC[Øw	normale Schriftart (Pica)
shorp2	15	ESC[2w	Elite ein
shorp1	16	ESC[1w	Elite aus
shorp4	17	ESC[4w	Schmalschrift (Condensed) ein
shorp3	18	ESC[3w	Schmalschrift aus
shorp6	19	ESC[6w	Breitschrift ein
shorp5	2Ø	ESC[5w	Breitschrift aus

den6	21	ESC[6"z	Schattenschrift ein
den5	22	ESC[5"z	Schattenschrift aus
den4	23	ESC[4"z	doppelter Anschlag (Doublestrike) ein
den3	24	ESC[3"z	doppelter Anschlag aus
den2	25	ESC[2"z	NLQ-Schönschrift ein
den1	26	ESC[1"z	NLQ-Schönschrift aus
sus2	27	ESC[2v	Hochstellen (Superscript) ein
sus1	28	ESC[1v	Hochstellen aus
sus4	29	ESC[4v	Tiefstellen (Subscript) ein
sus3	3Ø	ESC[3v	Tiefstellen aus
susØ	31	ESC[Øv	normale Stellung
plu	32	ESCL	Teillinie aufwärts
pld	33	ESCK	Teillinie abwärts
fntØ	34	ESC(B	US-Zeichensatz
fnt1	35	ESC(R	französischer Zeichensatz
fnt2	36	ESC(K	deutscher Zeichensatz
fnt3	37	ESC(A	englischer Zeichensatz
fnt4	38	ESC(E	dänischer Zeichensatz I
fnt5	39	ESC(H	schwedischer Zeichensatz
fnt6	4Ø	ESC(Y	italienischer Zeichensatz
fnt7	41	ESC(Z	spanischer Zeichensatz
fnt8	42	ESC(J	japanischer Zeichensatz
fnt9	43	ESC(6	norwegischer Zeichensatz
fnt1Ø	44	ESC(7	dänischer Zeichensatz II
prop2	45	ESC[2p	Proportionalschrift ein
prop1	46	ESC[1p	Proportionalschrift aus
propØ	47	ESC[Øp	Proportionalschrift löschen
tss	48	ESC[n E	Zeichenabstand n für Proportionalschrift
jfy5	49	ESC[5 F	linksbündig drucken
jfy7	5Ø	ESC[7 F	rechtsbündig drucken
jfy6	51	ESC[6 F	Blocksatz ein
jfyØ	52	ESC[Ø F	Blocksatz aus
jfy3	53	ESC[3 F	Briefformat
jfy1	54	ESC[1 F	zentriert drucken
verpØ	55	ESC[Øz	1/8" Zeilenabstand
verp1	56	ESC[1z	1/6" Zeilenabstand
slpp	57	ESC[nt	n Zeilen pro Blatt
perf	58	ESC[nq	Papier-Perforationsskip n
perfØ	59	ESC[Øq	kein Papier-Perforationsskip
lms	6Ø	ESC#9	linken Rand setzen
rms	61	ESC#Ø	rechten Rand setzen

tms	62	ESC#8	oberen Rand setzen
bms	63	ESC#2	unteren Rand setzen
stbm	64	ESC[Pn1;Pn2r	oberen Rand auf n1 und unteren auf n2 setzen
slrm	65	ESC[Pn1;Pn2s	linken Rand auf n1 und rechten auf n2 setzen
cam	66	ESC#3	alle Randdefinitionen löschen
hts	67	ESCH	Horizontaltabulator setzen
vts	68	ESCJ	Vertikaltabulator setzen
tbcØ	69	ESC[Øg	Horizontaltabulator löschen
tbc3	7Ø	ESC[3g	alle Horizontaltabulatoren löschen
tbc1	71	ESC[1g	Vertikaltabulator löschen
tbc4	72	ESC[4g	alle Vertikaltabulatoren löschen
tbcall	73	ESC#4	alle Tabulatoren löschen
tbsall	74	ESC#5	Standard-Tabulatoren setzen
extend	75	ESC[Pn"x	erweiterte Kommandos (?!)

Um Druckerkommandos zu verwenden, muß erstmal ein spezielles Request anstelle des IOStdReq initialisiert werden, der IOPrtCmdReq:

```
IOPrtCmdReq=RECORD
  message    : Message    ; Nachricht
  device     : DevicePtr  ; Zeiger auf Device
  unit       : UnitPtr    ; Zeiger auf Einheit
  command    : CARDINAL   ; Kommando
  flags      : IOFlagSet  ; IO-Flags
  error      : Error      ; Fehlermeldung
  prtCommand: CARDINAL    ; Druckerkommando
  parmØ      : [Ø..255]   ; Parameter Ø
  parm1      : [Ø..255]   ; Parameter 1
  parm2      : [Ø..255]   ; Parameter 2
  parm3      : [Ø..255]   ; Parameter 3
END;
```

Auch dieser Request wird mit CREATEEXTIO initialisiert und mit DELETEEXTIO entfernt.

Der einzige Befehl, der für das Senden von Druckerkommandos verfügbar ist, ist PRTCOMMAND:

command : prtCommand

prtCommand — Gewünschtes Druckerkommando (RIS..EXTEND, aus der Tabelle zu entnehmen)

p0–p3 — Gewünschte Parameter (aus der Tabelle zu entnehmen)

Rückgabewerte:

error	Evt. Fehlermeldung

8.5.3.3 Der Grafikdruck

Auch das Drucken von Grafiken ist beim Printer-Device vorgesehen, allerdings ohne komplizierte, eigene Routinen zu schreiben, sondern nur durch Angabe eines Device-Befehls mit den richtigen Parametern.

Für den Grafikdruck muß wieder ein spezieller Request mit CREATEEXTIO bereitgestellt werden (gelöscht werden kann er mit DELETEEXTIO), der IODRPReq:

```
IODRPReq=RECORD
  message  : Message     ; Nachricht
  device   : DevicePtr   ; Zeiger auf das Device
  unit     : UnitPtr     ; Zeiger auf die Einheit
  command  : CARDINAL    ; Kommando
  flags    : IOFlagSet   ; IO-Flags
  error    : Error       ; Fehlermeldung
  rastPort : RastPortPtr ; Zeiger auf den zu druckenden RastPort
  colorMap : ColorMapPtr ; Zeiger auf die zu druckende
                           Farbtabelle (für Farbdruck)
  modes    : LONGSET     ; Druckmodi
  srcX     : CARDINAL    ; x-Koordinate des Startpunktes
                           (Bildschirm!)
  srcY     : CARDINAL    ; y-Koordinate des Startpunktes
                           (Bildschirm!)
  srcWidth : CARDINAL    ; Breite des Bildes (Bildschirm!)
  srcHeight: CARDINAL    ; Höhe des Bildes (Bildschirm!)
  destCols : LONGINT     ; Breite des Ausdrucks
  destRows : LONGINT     ; Höhe des Ausdrucks
  special  : SpecialSet  ; Spezialflags
END;
```

modes In diesem Feld müssen die Darstellungsmodi des ViewPorts stehen, aber gibt es irgendeinen besonderen Grund, daß der hier stehende Wert vom Typ LONGSET sein muß??? Wie dem auch sei, da das nun so ist, müssen wir es wohl oder übel hinnehmen und uns Gedanken machen, wie denn die ViewModeSet-Werte in LONGCARDS umgerechnet werden. Da beide Typen gleich viel Speicherplatz benutzen, kann man mit CAST eine mehr oder minder bequeme Umwandlung vornehmen:

```
MeinRequest.modes:=CAST(LONGSET,ViewModeSet{hires});
```

Sie können natürlich auch einen beliebigen anderen Darstellungsmodus im Feld modes angeben.

srcX — Dieses Feld gibt die x-Koordinate der linken, oberen Ecke des zu druckenden Bildausschnittes an.

srcY — Hier muß die y-Koordinate der linken, oberen Ecke des zu druckenden Ausschnittes angegeben werden.

srcWidth — In diesem Feld muß die Breite des zu druckenden Bildausschnittes angegeben werden.

srcHeight — Hier muß die Höhe des zu druckenden Bildausschnittes angegeben werden.

destCols — Dieses Feld muß die Breite des Ausdrucks enthalten. Ist destCols größer als 0, wird der angegebene Wert für die Breite benutzt. Ist es gleich 0, so wird die maximale Breite des Papiers genutzt und ist der Inhalt dieses Feldes kleiner als 0, so kann der Ausdruck verkleinert oder vergrößert werden. Vorraussetzung für die Verkleinerung oder Vergrößerung ist, da der Inhalt des Feldes destRows größer als 0 ist. Der Vergrößerungsfaktor errechnet sich aus

```
destCols
---------- = Faktor
destRows
```

destRows — In diesem Feld wird die Höhe des Ausdrucks angegeben. Siehe auch destCols für weitere Informationen.

special — Hier können Sie einige Spezialmodi für den Ausdruck angeben. Die Werte der Menge SpecialSet sind zulässig:

milCols — Der Wert in destCols wird nicht in Punkten, sondern in 1/1000stel Zoll angegeben.

milRows — Der Wert in destRows wird nicht in Punkten, sondern in 1/1000stel Zoll angegeben.

fullCols — Es wird in der maximal möglichen Breite gedruckt.

fullRows — Es wird in der maximal möglichen Höhe gedruckt.

fracCols — Die Breite des Bildes ist gleich dem Maximum der zu bedruckenden Fläche geteilt durch destCols.

fracRows — Die Höhe des Bildes ist gleich dem Maximum der zu bedruckenden Fläche geteilt durch destRows.

center	Der Ausdruck wird zentriert.
aspect	Es wird im korrekten Seitenverhältnis gedruckt.
density1	Es wird in der niedrigsten Auflösung gedruckt.
density2	Es wird in der zweitniedrigsten Auflösung gedruckt.
density3	Es wird in der zweithöchsten Auflösung gedruckt.
density4	Es wird in der höchsten Auflösung gedruckt.

Der Befehl zum Drucken eines RastPorts heißt DUMPRPORT:

command : dumpRPort

rastPort	Hier muß ein Zeiger auf den zu druckenden RastPort stehen.
colorMap	In diesem Feld muß ein Zeiger auf die zu druckende ColorMap stehen.
modes	Hier müssen die ViewPort-Modi des Screens untergebracht sein.
srcX	Dieses Feld muß die x-Koordinate der linken, oberen Ecke des zu druckenden Ausschnittes enthalten.
srcY	In diesem Feld muß die y-Koordinate der linken, oberen Ecke des zu druckenden Ausschnittes stehen.
srcWidth	Hier muß die Breite des zu druckenden Ausschnittes stehen.
srcHeight	Dieses Feld muß die Höhe des zu druckenden Ausschnittes enthalten.
destCol	Hier muß die Breite des Ausdrucks stehen.
destRows	In diesem Feld muß die Höhe des Ausdrucks stehen.
special	Hier können Sie eventuelle Spezialmodi für den Grafikdruck angeben.

Rückgabewerte:

error	Evt. Fehlermeldung

Und nun zu einem Beispielprogramm, PicPrint, das eigentlich nur eine erweiterte Version des IFF-Laders aus dem GRAPHICS-Kapitel ist: Unkomprimierte IFF-Bilder werden geladen, angezeigt und anschließend ausgedruckt – auch farbig, wenn Ihr Drucker das erlaubt:

```
MODULE PicPrint;

FROM SYSTEM       IMPORT ADR,ADDRESS,LONGSET;
FROM Intuition    IMPORT NewScreen,ScreenPtr,OpenScreen,CloseScreen,customScreen;
FROM Graphics     IMPORT RastPortPtr,BitMapPtr,SetRGB4,ViewModes,ViewModeSet;
FROM Dos          IMPORT Open,Close,Read,Write,FileHandlePtr,Delay,Exit;
FROM Exec         IMPORT AllocMem,FreeMem,MemReqs,MemReqSet,OpenDevice,
                         CloseDevice,MsgPortPtr,DoIO;
FROM ExecSupport IMPORT CreateExtIO,DeleteExtIO,CreatePort,DeletePort;
FROM Printer      IMPORT IODRPReqPtr,SpecialSet,Special,dumpRPort;
FROM Arguments    IMPORT GetArg;
FROM Terminal     IMPORT WriteString,WriteLn;
FROM Strings      IMPORT Compare;

CONST
    dppv = "DPPV";
    bmhd = "BMHD";
    cmap = "CMAP";
    crng = "CRNG";
    body = "BODY";
    camg = "CAMG";
    ccrt = "CCRT";

VAR MeinScreen : ScreenPtr;
    ScreenDaten: NewScreen;
    RP         : RastPortPtr;
    BMap        : BitMapPtr;
    Datei       : FileHandlePtr;
    i,j         : INTEGER;
    a           : LONGINT;
    R,G,B       : [Ø..255];
    Abfall      : ADDRESS;
    Ende        : BOOLEAN;
    Use         : LONGINT;
    text        : ARRAY[Ø..3] OF CHAR;
    Buffer      : ARRAY[Ø..5] OF ADDRESS;
    Dateiname   : ARRAY[Ø..5Ø] OF CHAR;
    MeinPort    : MsgPortPtr;
    DmpReq      : IODRPReqPtr;
```

```
BEGIN
  GetArg(1,Dateiname,i); (* Argument (Dateinamen) holen *)
(* Screen-"Gerüst" erzeugen, das später ausgefüllt wird *)
  WITH ScreenDaten DO
    leftEdge:=Ø; topEdge:=Ø; width:=Ø; height:=Ø; depth:=Ø;
    detailPen:=Ø; blockPen:=1; viewModes:=ViewModeSet{};
    type:=customScreen;
    font:=NIL; gadgets:=NIL; defaultTitle:=NIL;
    customBitMap:=NIL;
  END;

  Datei:=Open(ADR(Dateiname),1ØØ5); (* Datei im "modeOldfile" öffnen *)
  IF Datei=Ø THEN
    WriteLn; WriteString("Keine solche Datei gefunden!");
    Exit(1ØØØ);
  END;

  Abfall:=AllocMem(1Ø8,MemReqSet{chip,memClear}); (* 1Ø8 Byte für uninter-
              essante Daten allozieren *)
  Ende:=FALSE;

  Use:=Read(Datei,Abfall,12); (* Datei-Header überlesen *)

  WHILE Ende=FALSE DO
    Use:=Read(Datei,ADR(text),4);                    (* Kennung laden *)

    Use:=Compare(text,Ø,4,bmhd,TRUE);       (* BMHD? *)
    IF Use=Ø THEN            (* jawohl! *)
      Use:=Read(Datei,Abfall,4);            (* Schund überlesen *)
      Use:=Read(Datei,ADR(ScreenDaten.width),2);  (* Breite einlesen *)
      Use:=Read(Datei,ADR(ScreenDaten.height),2); (* Höhe einlesen *)
      Use:=Read(Datei,Abfall,3);      (* 3 Byte überlesen *)
      Use:=Read(Datei,ADR(ScreenDaten.depth),2);  (* Anzahl Planes einlesen *)
      Use:=Read(Datei,Abfall,11);     (* 11 Byte überlesen *)
      IF ScreenDaten.width > 32Ø THEN                (* Bild breiter als 32Ø? *)
        INCL(ScreenDaten.viewModes,hires);   (* muß HIRES sein!*)
      END;
      IF ScreenDaten.height > 256 THEN     (* Bild höher als 256? *)
        INCL(ScreenDaten.viewModes,lace);   (* muß LACE sein! *)
      END;
      MeinScreen:=OpenScreen(ScreenDaten);          (* Screen öffnen *)
      RP:=ADR(MeinScreen^.rastPort);     (* RastPort definieren *)
      BMap:=RP^.bitMap;        (* BitMap definieren *)
      FOR i:=Ø TO ScreenDaten.depth-1 DO    (* Zwischenspeicher für die *)
       Buffer[i]:=BMap^.planes[i];     (* Adresse der Bitplanes *)
      END;
    END;
```

```
  Use:=Compare(text,Ø,4,cmap,TRUE);             (* ist Kennung CMAP? *)
  IF Use=Ø THEN       (* ja! *)
    Use:=Read(Datei,ADR(a),4);    (* Anzahl Farben lesen *)
    a:=a DIV 3;       (* durch 3 teilen *)
    FOR i:=Ø TO a-1 DO
      Use:=Read(Datei,ADR(R),1);    (* Rotkomponente einlesen *)
      Use:=Read(Datei,ADR(G),1);    (* Grünkomponente einlesen *)
      Use:=Read(Datei,ADR(B),1);    (* Blaukomponente einlesen *)
      SetRGB4(ADR(MeinScreen^.viewPort),
      i,R DIV 16,G DIV 16,B DIV 16); (* Farben setzen! *)
    END;
  END;

  Use:=Compare(text,Ø,4,body,TRUE);    (* ist Kennung BODY? *)
  IF Use=Ø THEN       (* ja! *)
    Use:=Read(Datei,Abfall,4);    (* Abfall überlesen! *)
    FOR i:=Ø TO ScreenDaten.height-1 DO          (* Bitmap einlesen *)
      FOR j:=Ø TO ScreenDaten.depth-1 DO
        Use:=Read(Datei,Buffer[j]+(ScreenDaten.width DIV 8 * i),
                ScreenDaten.width DIV 8);
      END;
    END;
    Ende:=TRUE;
  END;

  Use:=Compare(text,Ø,4,crng,TRUE);             (* CRNG? *)
  IF Use=Ø THEN       (* uninteressant! *)
    Use:=Read(Datei,Abfall,12);
  END;

  Use:=Compare(text,Ø,4,dppv,TRUE);             (* DPPV? *)
  IF Use=Ø THEN       (* uninteressant! *)
    Use:=Read(Datei,Abfall,1Ø8);
  END;

  Use:=Compare(text,Ø,4,camg,TRUE);             (* CAMG? *)
  IF Use=Ø THEN       (* uninteressant! *)
    Use:=Read(Datei,Abfall,8);
  END;

  Use:=Compare(text,Ø,4,ccrt,TRUE);             (* CCRT? *)
  IF Use=Ø THEN       (* uninteressant! *)
    Use:=Read(Datei,Abfall,18);
  END;
END; (* von WHILE *)

Close(Datei);       (* Schließen *)
```

```
  MeinPort:=CreatePort(NIL,Ø);      (* Port generieren *)
  DmpReq:=CreateExtIO(MeinPort,SIZE(DmpReq^));   (* Request generieren *)

  OpenDevice(ADR("printer.device"),Ø,DmpReq,LONGSET{}); (* Device öffnen *)

  (* Hier wird der Request auf den Ausdruck angepaßt *)
  WITH DmpReq^ DO
    command:=dumpRPort; rastPort:=RP;
    colorMap:=MeinScreen^.viewPort.colorMap;
    srcX:=Ø; srcY:=Ø; srcWidth:=ScreenDaten.width; destCols:=Ø;
    destRows:=Ø;
    srcHeight:=ScreenDaten.height;
    special:=SpecialSet{fullCols,aspect};
  END;

  DoIO(DmpReq);       (* OK — GO!!! *)

  CloseScreen(MeinScreen);
  FreeMem(Abfall,1Ø8);

  CloseDevice(DmpReq);      (* Device schließen *)
  DeleteExtIO(DmpReq);       (* Request beseitigen *)
  DeletePort(MeinPort);       (* Port beseitigen *)
END PicPrint.
```

8.5.3.4 Fehlermeldungen

Auch das Printer-Device kennt seine eigenen Fehlermeldungen, die eventuell im Feld error zurückgegeben werden und in der Aufzählung Error aus Printer verzeichnet sind:

e0 — Dummy-Wert, der absolut nichts aussagt und garantiert nie vorkommt.

cancel — Der Benutzer provozierte einen Druckabbruch.

notGraphics — Sie versuchten, auf einem nicht grafikfähigen Drucker eine Grafik zu drucken.

invertHam — HAM-Ausdruck konnte nicht invertiert werden.

badDimension — Dimensionen des Ausdrucks sind unzulässig.

imensionOvflow — Dimensionen des Ausdrucks sind zu groß.

internalMemory — Nicht genug Speicher für interne Variablen.

bufferMemory — Nicht genug Speicher für den Druckerspeicher.

8.5.4 Das Console-Device

Das Console-Device dient vor allem der komfortablen Tastatursteuerung, mit ihm können Sie Daten von der Tastatur lesen, Kontrollstrings senden oder sogar eigene Tastaturbelegungen herstellen, doch dazu kommen wir später.

Wie das Console-Device zu öffnen ist, dürften Sie eigentlich schon erraten haben: genauso wie die anderen Devices. Vorher müssen Sie natürlich einen Nachrichtenport kreieren, aber das ist ja »Schnee von gestern«. Lediglich die Art des Requests dürfte noch unklar sein; des Rätsels Lösung ist ein ganz normaler IOStdReq-Request.

Eines ist aber sehr wichtig: vor dem Öffnen des Console-Device muß ein Zeiger auf das Window, für das dieses Console-Device gelten soll, im Feld data übergeben werden. Im Feld length muß die Größe des Window-Records stehen. Danach erst sollten Sie den OPENDEVICE-Befehl anwenden (als Name ist »console.device« anzugeben).

Folgende Standardkommandos stehen zur Verfügung:

```
read(#2)Tastatureingabe lesen
write(#3)String senden
clear(#4)Konsolenspeicher löschen
```

Die vier devicespezifischen Befehle beziehen sich ausschließlich auf die Handhabung eigener Tastaturtabellen, weshalb sie auch später aufgeführt werden sollen.

8.5.4.1 Die Standard-Befehle

Die Funktion des READ-Befehls dürfte klar sein. Er liest eine angegebene Anzahl an Tastatureingaben (umgangssprachlich auch Tastendrücke genannt) in einen angegebenen Buffer ein. Beachten Sie bitte, daß diese NICHT auf dem Bildschirm ausgegeben werden, dafür ist der WRITE-Befehl zuständig. So müssen die Felder des IOStdReq-Records vor dem DOIO-Aufruf gefüllt sein:

command :read (oder #2)

data	Adresse eines Speicherbereichs, in dem die ASCII-Codes der gedrückten Tasten gespeichert werden.
length	Anzahl der zu lesenden Tastendrücke

Rückgabewerte:

keine

Etwas anders ist es da mit WRITE. WRITE gibt einen angegebenen Speicherbereich (meistens wird ein String benutzt) auf dem spezifizierten Fenster, das Sie vor dem OPENDEVICE-Aufruf angegeben haben, aus:

command : write (oder #3)

data	Adresse des auszugebenden Strings
length	Länge des auszugebenden Strings in Zeichen. Wird hier -1 angegeben, werden alle Zeichen des Strings auf dem Bildschirm, d.h. das spezifizierte Window, ausgegeben.

Rückgabewerte:

keine

Nun gibt es beim Console-Device, wie beim Printer-Device, eine ganze Reihe von Kontrollsequenzen, die mit WRITE ausgegeben oder mit READ eingelesen werden können. Diese sind:

Sequenz	Wirkung
Ø7H	Bildschirmblinken
Ø8H	Rückschritt ohne Löschen des links vom Cursor liegenden Zeichens
ØAH	Cursor wird um eine Zeile nach unten bewegt
ØBH	Cursor wird um eine Zeile nach oben bewegt
ØCH	Das Fenster des Console-Devices wird gelöscht
ØDH	Cursor wird an den Anfang der Zeile gesetzt
ØEH	SHIFT-Taste aktivieren
ØFH	SHIFT-Taste deaktivieren
Ø1BH	ESC-Taste, kann eine Kontrollsequenz einleiten
Ø9BH	CSI-Code, leitet eine Kontrollsequenz ein
Ø1BH Ø63H	Console-Device in den Ursprungszustand versetzen
Ø9BH n Ø4ØH	n Leerzeichen einfügen; wird n weggelassen, wird nur ein Leerzeichen eingefügt
Ø9BH n Ø41H	Cursor um n Stellen nach oben schieben; wird n weggelassen, wird der Cursor nur um eine Stelle verschoben.
Ø9BH n Ø42H	Cursor um n Stellen nach unten schieben; ohne n wird der Cursor eine Stelle nach unten geschoben
Ø9BH n Ø43H	Cursor um n Stellen nach rechts verschieben; ohne n wird der Cursor eine Stelle nach rechts geschoben
Ø9BH n Ø44H	Cursor um n Stellen nach links verschieben; ohne n wird der Cursor eine Stelle nach links geschoben
Ø9BH n Ø45H	Cursor um n Zeilen nach unten bewegen und an den Anfang der nächsten Zeile setzen
Ø9BH n Ø46H	Cursor um n Zeilen nach oben bewegen und an den Zeilenanfang setzen

Ø9BH n Ø3BH m Ø48H	Cursor an die Position (n;m) setzen. Wird m nicht angegeben (Cursor bleibt in der gleichen Zeile), darf auch Ø3BH nicht angegeben werden!
Ø9BH Ø4AH	Fenster von der aktuellen Cursorposition bis zum Ende löschen
Ø9BH Ø4BH	Zeile bis zum Ende löschen
Ø9BH Ø4CH	Zeile über der, in der der Cursor steht, einfügen
Ø9BH Ø4DH	Zeile über der, in der der Cursor steht, löschen und die anderen Zeilen aufrücken
Ø9BH n Ø5ØH	n Zeichen rechts vom Cursor löschen, oder das Zeichen unter dem Cursor, wenn n fehlt
Ø9BH n Ø53H	scrollt den Bildschirm n Zeilen nach oben (die unteren n Zeilen werden dabei gelöscht)
Ø9BH n Ø54H	scrollt den Bildschirm n Zeilen nach unten, wobei die oberen n Zeilen gelöscht werden
Ø9BH Ø32H Ø3ØH Ø68H	der Code ØAH (Linefeed, s.o.) wird als ØAH und ØDH angenommen.
Ø9BH Ø32H Ø3ØH Ø6CH	der Code ØAH wird wieder als solcher angenommen (mit der vorherigen Sequenz kann er umgestellt werden)
Ø9BH Ø36H Ø6EH	das Console-Device sendet die aktuelle Cursorposition in der Form <Ø9BH n Ø3BH m>. Diese Position kann mit dem Kommando READ eingelesen und ausgewertet werden.

Nun kommt ein ganz spezielles Kommando, das es erlaubt, verschiedene Schriftstilarten und -farben anzugeben:

09BH Stil 03BH Vordergrundfarbe 03BH Hintergrundfarbe 06DH

»Stil« ist hierbei die gewünschte Schriftstilart:

1.Ø3ØH: normale Schriftart

2.Ø31H: Fettschrift

3.Ø33H: Kursivschrift

4.Ø34H: Unterstreichen

5.Ø37H: Inversschrift

»Vordergrundfarbe« gibt die gewünschte Vordergrundfarbe an, abhängig von der Anzahl der Bitplanes des Screens. Maximal sind acht verschiedene Farben möglich:

Ø33H Ø3ØH: Farbe 1
Ø33H Ø31H: Farbe 2
Ø33H Ø32H: Farbe 3
Ø33H Ø33H: Farbe 4
Ø33H Ø34H: Farbe 5

```
Ø33H Ø35H: Farbe 6
Ø33H Ø36H: Farbe 7
Ø33H Ø37H: Farbe 8
```

Mit »Hintergrundfarbe« können Sie eine von maximal acht Hintergrundfarben wählen:

```
Ø34H Ø3ØH: Farbe 1
Ø34H Ø31H: Farbe 2
Ø34H Ø32H: Farbe 3
Ø34H Ø33H: Farbe 4
Ø34H Ø34H: Farbe 5
Ø34H Ø35H: Farbe 6
Ø34H Ø36H: Farbe 7
Ø34H Ø37H: Farbe 8
```

Jetzt kommen noch einige Amiga-spezifische Console-Device Steuerkommandos:

```
Ø9BH n Ø74H            Arbeitsbereich für das Console-Device auf n
                       Zeilen begrenzen. Möchten Sie, daß das
                       Console-Device die Größe des Arbeitsbereiches
                       wieder selbst festlegt, genügt es, n wegzulassen.
Ø9BH n Ø75H            setzt die Anzahl der Zeichen, die in eine Zeile
                       passen sollen auf n
Ø9BH n Ø78H            setzt den Anfang des Textbereiches auf n Punkte
                       rechts vom linken Fensterrand
Ø9BH n Ø79H            setzt den Anfang des Textbereiches auf n Punkte
                       unter dem oberen Fensterrand
Ø9BH Ø3ØH Ø2ØH Ø7ØH    macht den Cursor unsichtbar
Ø9BH Ø2ØH Ø7ØH         macht den Cursor wieder sichtbar
Ø9BH Ø3ØH Ø2ØH Ø71H    liefert die Größe des Console-Device-Fensters zurück.
                       Diese Größe kann mit dem READ-Befehl eingelesen und
                       ausgewertet werden. Sie hat folgende Form:

                       Ø9BH Ø31H Ø3BH Ø31H Ø3BH y2 Ø3BH x2 Ø2ØH Ø72H
```

y2 Ist die y-Koordinate der rechten, unteren Ecke und »x2« die x-Koordinate dieser Ecke. Beide Koordinaten benutzen die Einheit Zeichen und benötigen zwei ASCII-Codes: der erste steht für die Zehner-, der andere für die Einerstelle.

8.5.4.2 RAW-Events

Reichen Ihnen die Informationen, die Ihnen das READ-Kommando über Tastendrücke liefert, nicht aus, können Sie auf die sogenannten RAW-Events zurückgreifen. Bei diesen bekommen Sie eine Meldung, wenn ein bestimmtes Ereignis eingetreten ist. Möchten Sie RAW Events benutzen, so müssen Sie erst eine Kontrollsequenz mit den Ereignissen, bei deren Eintreten das Console-Device Meldung machen soll, senden:

09BH Ereignismeldungen, durch 03BH voneinander getrennt sein müssen 07BH

Die einzelnen Codes der Ereignisse sind:

030H Reserviert

031H Meldung beim Drücken und Loslassen einer Taste

032H Meldung beim Drücken einer Maustaste

033H Meldung beim Aktivieren eines Windows

034H Meldung beim Bewegen der Maus

035H Unbenutzt

036H Meldung bei Timer-Events

037H Meldung beim Anklicken eines Gadgets

038H Meldung beim Loslassen eines Gadgets

039H Meldung bei der Darstellung eines Requesters

03AH Meldung beim Anwählen eines Menüs

03BH Meldung beim Anklicken des Schließsymbols

03CH Meldung bei veränderter Fenstergröße

03DH Meldung bei einem Window-Refresh

03EH Meldung bei geänderten Preferences

03FH Meldung beim Herausnehmen einer Diskette

040H Meldung beim Einlegen einer Diskette

Erinnert das nicht irgendwie an IDCMP-Flags? Richtig, es müssen auch die entsprechenden IDCMP-Flags gesetzt sein, damit die RAW-Events funktionieren.

Was passiert aber nun, wenn ein Ereignis, das eine Meldung auslösen soll, eintritt? Es wird eine Kontrollsequenz abgeschickt, die mit READ aufgefangen werden kann:

<CSI><class>;<subclass>;<keycode>;<qualifiers>;<x>;<y>;<seconds>;<microseconds>

Die Semikola sowie das »¦«-Zeichen sind natürlich durch die entsprechenden ASCII-Codes zu ersetzen, doch ich habe diese Notation gebraucht, um den Kontrollstring etwas übersichtlicher erscheinen zu lassen.

<CSI> Dieser Parameter entspricht dem Wert 09BH.

<class> Code der eingetroffenen Meldung.

<subclass> Ist normalerweise 0, nur wenn die Maus im Port 2 steckt, ist dieser Wert 1.

<keycode> Enthält den Tastaturcode der gedrückten oder losgelassenen Taste (s.u.)

<qualifiers> Enthält den Code einer evt. zusätzlich gedrückten Sondertaste:

1	linke <SHIFT>-Taste
2	rechte <SHIFT>-Taste
4	<CapsLock>-Taste
8	<Control>-(<CTRL>-)Taste
16	linke <ALT>-Taste
32	rechte <ALT>-Taste
64	linke <Amiga>-Taste
128	rechte <Amiga>-Taste
256	Taste liegt auf dem Zehnerblock
512	Repeatfunktion ist eingeschaltet (Taste wird laufend wiederholt)
1024	unbenutzt
2048	das Ereignis gilt nur für das aktive Fenster
4096	linker Mausknopf
8192	unbenutzt (evt. mittlerer Mausknopf)
16384	rechter Mausknopf
32768	Mausbewegung

(Bei mehreren Werten werden diese zueinander addiert)

<x> Anzahl der Punkte, um die die Maus in x-Richtung bewegt wurde, falls sie bewegt wurde.

<y> Anzahl der Punkte, um die die Maus in y-Richtung bewegt wurde, sollte sie überhaupt bewegt worden sein.

<seconds> Aktuelle Systemzeit (Sekunden).

<microseconds> Aktuelle Systemzeit (Mikrosekunden).

Wurde ein Gadget betätigt oder losgelassen, so muß man »<x>« um 16 Bit nach links SHIFTen und »<y>« addieren, um die Adresse des entsprechenden Gadgets zu erhalten.)

Hier nun die versprochene Tabelle der Tastencodes (deutsche Tastaturbelegung):

Nummer	Taste (ohne SHIFT)	Taste (mit SHIFT)
ØØØH	`	~
ØØ1H	1	!
ØØ2H	2	"
ØØ3H	3	#
ØØ4H	4	$
ØØ5H	5	%
ØØ6H	6	&
ØØ7H	7	/
ØØ8H	8	(
ØØ9H	9	)
ØØAH	Ø	=
ØØBH	ß	?
ØØCH	(Accent aigu, Akut)	(Accent grave, Gravis)
ØØDH	\	¦
ØØEH	undefiniert	undefiniert
ØØFH	Ø (Zehnerblock)	Ø (Zehnerblock)
Ø1ØH	q	Q
Ø11H	w	W
Ø12H	e	E
Ø13H	r	R
Ø14H	t	T
Ø15H	z	Z
Ø16H	u	U
Ø17H	i	I
Ø18H	o	O
Ø19H	p	P
Ø1AH	ü	Ü
Ø1BH	+	*
Ø1CH	undefiniert	undefiniert
Ø1DH	1 (Zehnerblock)	1 (Zehnerblock)
Ø1EH	2 (Zehnerblock)	2 (Zehnerblock)
Ø1FH	3 (Zehnerblock)	3 (Zehnerblock)
Ø2ØH	a	A
Ø21H	s	S
Ø22H	d	D
Ø23H	f	F
Ø24H	g	G
Ø25H	h	H
Ø26H	j	J
Ø27H	k	K
Ø28H	l	L
Ø29H	ö	Ö
Ø2AH	ä	Ä

Ø2BH	#	
Ø2CH	undefiniert	undefiniert
Ø2DH	4 (Zehnerblock)	4 (Zehnerblock)
Ø2EH	5 (Zehnerblock)	5 (Zehnerblock)
Ø2FH	6 (Zehnerblock)	6 (Zehnerblock)
Ø3ØH	<	>
Ø31H	y	Y
Ø32H	x	X
Ø33H	c	C
Ø34H	v	V
Ø35H	b	B
Ø36H	n	N
Ø37H	m	M
Ø38H	,	;
Ø39H	.	:
Ø3AH	—	_
Ø3BH	undefiniert	undefiniert
Ø3CH	. (Zehnerblock)	. (Zehnerblock)
Ø3DH	7 (Zehnerblock)	7 (Zehnerblock)
Ø3EH	8 (Zehnerblock)	8 (Zehnerblock)
Ø3FH	9 (Zehnerblock)	9 (Zehnerblock)
Ø4ØH	(Leertaste)	(Leertaste)
Ø41H	(Backspace)	(Backspace)
Ø42H	(Tabulatortaste)	(Tabulatortaste)
Ø43H	(Enter-Taste)	(Enter-Taste)
Ø44H	(Return-Taste)	(Return-Taste)
Ø45H	(ESC-Taste)	(ESC-Taste)
Ø46H	(DEL-Taste)	(DEL-Taste)
Ø47H	undefiniert	undefiniert
Ø48H	undefiniert	undefiniert
Ø49H	undefiniert	undefiniert
Ø4AH	— (Zehnerblock)	— (Zehnerblock)
Ø4BH	undefiniert	undefiniert
Ø4CH	(Cursor hoch)	(Cursor hoch)
Ø4DH	(Cursor runter)	(Cursor runter)
Ø4EH	(Cursor rechts)	(Cursor rechts)
Ø4FH	(Cursor links)	(Cursor links)
Ø5ØH	(F1)	(F1)
Ø51H	(F2)	(F2)
Ø52H	(F3)	(F3)
Ø53H	(F4)	(F4)
Ø54H	(F5)	(F5)
Ø55H	(F6)	(F6)
Ø56H	(F7)	(F7)

Ø57H	(F8)	(F8)
Ø58H	(F9)	(F9)
Ø59H	(F1Ø)	(F1Ø)
Ø5AH	undefiniert	undefiniert
Ø5BH	undefiniert	undefiniert
Ø5CH	undefiniert	undefiniert
Ø5DH	undefiniert	undefiniert
Ø5EH	undefiniert	undefiniert
Ø5FH	(HELP-Taste)	(HELP-Taste)
Ø6ØH	(linke SHIFT-Taste)	(linke SHIFT-Taste)
Ø61H	(rechte SHIFT-Taste)	(rechte SHIFT-Taste)
Ø62H	(CapsLock-Taste)	(CapsLock-Taste)
Ø63H	(CTRL-Taste)	(CTRL-Taste)
Ø64H	(linke ALT-Taste)	(linke ALT-Taste)
Ø65H	(rechte ALT-Taste)	(rechte ALT-Taste)
Ø66H	(linke Amiga-Taste)	(linke Amiga-Taste)
Ø67H	(rechte Amiga-Taste)	(rechte Amiga-Taste)
Ø68H	(linker Mausknopf)	(linker Mausknopf)
Ø69H	(rechter Mausknopf)	(rechter Mausknopf)
Ø6AH	undefiniert	undefiniert
Ø6BH	undefiniert	undefiniert
Ø6CH	undefiniert	undefiniert
Ø6DH	undefiniert	undefiniert
Ø6EH	undefiniert	undefiniert
Ø6FH	undefiniert	undefiniert
Ø7ØH-Ø7FH	undefiniert	undefiniert
Ø8ØH-ØF8H	entsprechen eigentlich den Codes ØØØH-Ø7FH, nur daß die Taste losgelassen wurde	
ØF9H	illegaler Code	
ØFAH	Tastaturspeicher läuft über	
ØFBH	undefiniert, reserviert für einen Tastaturprozessorabsturz	
ØFCH	Tastaturselbsttest verlief fehlerhaft	
ØFDH	Tasten wurden während des Bootvorgangs gedrückt	
ØFEH	falls Tasten bei einem Bootvorgang gedrückt wurden, so wurden diese gesendet und die Tastatur aufnahmeklar gemacht	
ØFFH	(Mausbewegung, falls gewählt, sonst undefiniert)	

Um den Speicher des Console-Devices zu löschen, genügt es, den Befehl CLEAR anzuwenden:

command: clear (oder #5)

Rückgabewerte:

keine

Mit diesem Kapitel (nicht aber mit dem Console-Device) dürften wir fertig sein, was auch der Grund für das nun kommende Beispielprogramm ist. Dabei haben Sie 500 Intuiticks (ca. 10 Sekunden) Zeit, Ihren Namen einzugeben, bevor das Programm Sie entsprechend begrüßt:

```
MODULE ConsoleDemo;

FROM SYSTEM        IMPORT LONGSET,ADR;
FROM Intuition     IMPORT Window,WindowPtr;
FROM Exec          IMPORT MsgPortPtr,IOStdReqPtr,OpenDevice,IOStdReq,DoIO,
                          CloseDevice,read,write;
FROM ExecSupport IMPORT CreatePort,CreateStdIO,DeleteStdIO,DeletePort;
FROM Dos           IMPORT Delay;
FROM Strings       IMPORT Insert,Length;
FROM Windows       IMPORT OpenWindow,CloseWindow,WinGad,WinGadSet;

VAR dport   : MsgPortPtr;
    dreq    : IOStdReqPtr;
    Fenster: WindowPtr;
    Win     : Window;
    S1,S2   : ARRAY[Ø..5Ø] OF CHAR;

BEGIN
  (* zum Betrieb des Devices nötiges Fenster öffnen *)

  OpenWindow(Fenster,Ø,Ø,64Ø,3Ø,"Eingabefenster",WinGadSet{- moving,arranging});

  dport:=CreatePort(NIL,Ø); (* Port einrichten *)
  dreq:=CreateStdIO(dport); (* Request einrichten *)

  dreq^.data:=Fenster;      (* Adresse des zu benutzenden Fensters übergeben *)
  dreq^.length:=SIZE(Win);  (* Länge des Window-Records übergeben *)

  (* Device öffnen *)
  OpenDevice(ADR("console.device"),Ø,dreq,LONGSET{}); (* Device öffnen *)

  (* Ausgabetext vorbereiten, "3;33;4Øm" entspricht den ASCII-Codes für
     Schrägschrift, orangene Zeichenfarbe und normale Hintergrundfarbe. Daran
     wird der eigentliche Text und das Kommandosequenz-Zeichen CSI (Ø9BH) an-
     gehängt. *)
  S1:=" 3;33;4ØmWie heißen Sie? ";
  S1[Ø]:=CHAR(Ø9BH);
```

```
  dreq^.command:=write;      (* Kommando WRITE *)
  dreq^.data:=ADR(S1);       (* Adresse des auszugebenden Textes *)
  dreq^.length:=26;      (* Länge des Textes in Zeichen *)
  DoIO(dreq);        (* GO!! *)

  Delay(5ØØ);        (* Eingabezeit *)

  dreq^.command:=read;       (* Kommando READ, um Daten einzulesen *)
  dreq^.data:=ADR(S2);       (* Adresse des Einlesestrings *)
  dreq^.length:=1Ø;      (* maximale Anzahl einzulesender Zeichen *)
  DoIO(dreq);        (* ausführen! *)

  S1:=" Ø;31;4ØmHallo, ";   (* Antwortstring vorbereiten *)
  S1±Ø[:=CHAR(Ø9BH);
  Insert(S2,Ø,S1);      (* Namen einfügen *)

  dreq^.command:=write;      (* WRITE um die Begrüßung auszugeben *)
  dreq^.data:=ADR(S2);       (* Begrüßungsstring *)
  dreq^.length:=Length(S2); (* Länge dieses Strings *)
  DoIO(dreq);        (* ausführen! *)

  Delay(25Ø);        (* warte 'ne Weile ... *)

  (* Arbeit beendet, hinterlasse geordneten Zustand! *)
  CloseDevice(dreq);     (* Device schließen *)
  CloseWindow(Fenster);     (* Fenster schließen *)
  DeleteStdIO(dreq);     (* Request vernichten *)
  DeletePort(dport);     (* Port vernichten *)
END ConsoleDemo.
```

8.5.5 Das Timer-Device

Das Timer-Device ist die Uhr des Amiga. Davon gibt es zwei Stück, den VBlank- und den CIA-Timer. Beide Timer haben so ihre Vor- und Nachteile. Zu den Nachteilen gehört sicherlich, daß beide »Uhren« mehr oder minder ungenau gehen, was aber durch das multitaskingfähige Betriebssystem des Amiga bedingt ist. Der VBlank-Timer geht recht genau, die kleinste Einheit, die er verwalten kann, sind aber 50stel Sekunden, was für manche Anwendungen zu grob sein kann. Der CIA-Timer ist da schon manchmal interessanter, kann er doch sogar Mikrosekunden (millionstel Sekunden) messen. Doch seine Genauigkeit läßt bereits nach einer Sekunde zu wünschen übrig.

Den Request, den das Timer-Device benötigt, hat wieder eine spezielle Form:

```
TimeRequest=RECORD
  node: IORequest ; der ganz normale IORequest
  time: TimeVal   ; Zeitstruktur
END;
```

Der TimeVal-Record hat folgende Form:

```
TimeVal=RECORD
  secs : LONGCARD ; Sekunden
  micro: LONGCARD ; Mikrosekunden
END;
```

Dieser Request kann mit CREATEEXTIO initialisiert und mit DELETEEXTIO gelöscht werden.

Beim OPENDEVICE-Befehl hat der Parameter für die Einheit (Unit) eine Bedeutung, er muß angeben, welchen Timer Sie benutzen möchten. Dafür existieren im Timer-Modul zwei Konstanten:

microHz (#0)	Der CIA-Timer soll benutzt werden.
vBlank (#1)	Der VBlank-Timer soll benutzt werden.

Es gibt drei devicespezifische Befehle für den Timer:

addRequest (nonstd+0)	Für eine bestimmte Zeit warten.
getSysTime (nonstd+1)	Aktuelle Systemzeit holen.
setSysTime (nonstd+2)	Systemzeit setzen.

Zu ADDREQUEST: dieser Befehl wartet eine bestimmte Zeit und springt dann ins Programm zurück. Die Dauer des Wartezustandes müssen Sie in einem TimeVal-Record festlegen, das Feld secs muß die Anzahl der Sekunden und das Feld micro die Anzahl der zu wartenden Mikrosekunden enthalten, wobei letzteres bei der Verwendung des VBlank-Timers sinnlos ist. Die Felder müssen wie folgt ausgefüllt sein:

command : addRequest

data	Adresse des zu benutzenden TimeVal-Records.

Rückgabewerte:

keine

Der Befehl GETSYSTIME holt die aktuelle Systemzeit, die aber, wie in vorherigen Kapiteln schon angedeutet, nicht in Tagen, Monaten und Jahren, sondern in Sekunden und Mikrosekunden seit dem 1.1.1978 gezählt wird:

command : getSysTime

Rückgabewerte:

time	TimeVal-Record, der die aktuelle Systemzeit enthält.

Möchten Sie die Systemzeit neu setzen, sollten Sie den Befehl SETSYSTIME anwenden:

command : setSysTime

data	Adresse des TimeVal-Records, dessen Inhalt die neue Systemzeit bilden soll.

Rückgabewerte:

keine

Nun stellt das Timer-Device neben den Device-Kommandos noch Befehle in Prozedurform zur Verfügung: ADDTIME, CMPTIME und SUBTIME. Alle drei Befehle brauchen einen Zeiger auf das Device, an den aber einfach heranzukommen ist:

DeviceZeiger := MeinTimerRequest^.node^.device;

DeviceZeiger muß hierbei selbstverständlich vom Typ DevicePtr sein. Doch nun zu den einzelnen Befehlen, angefangen mit ADDTIME. ADDTIME addiert zwei TimeVal-Records und speichert das Ergebnis im ersten ab:

AddTime (Timer , Val1 , Val2);

Timer	ADDRESS-Wert, der einen Zeiger auf das Timer-Device enthält.
Val1	TimeVal-Wert, der den ersten TimeVal-Record und später auch das Ergebnis der Addition enthält.
Val2	TimeVal-Record, dessen Inhalt zu »Val1« addiert wird.

Ähnlich wie ADDTIME wirkt der Befehl SUBTIME, der den zweiten TimeVal-Record vom ersten subtrahiert und das Ergebnis im letzteren sichert:

SubTime (Timer , Val1 , Val2);

Timer	ADDRESSe des Timer-Devices.
Val1	TimeValPtr, der den TimeVal-Record mit dem Subtrahenden und später auch den mit der Differenz zeigt.
Val2	TimeValPtr, dessen Record-Inhalt von »Val1« abgezogen wird.

Dieser Befehl ist äußerst nützlich, um z.B. die Zeit, die ein Programm braucht, zu messen. Man muß nur die Systemzeit am Programmstart und -ende abfragen und dann die erste Zeit von der zweiten abziehen, schon erhält man die vergangene Zeit.

Neben ADDTIME und SUBTIME gibt es noch den CMPTIME-Befehl, der zwei TimeVal-Records miteinander vergleicht:

Ergebnis := CmpTime (Timer , Val1 , Val2);

Timer	ADDRESSe des Timer-Devices
Val1	TimeValPtr, der auf den TimeVal-Record, der mit »Val2« verglichen werden soll, zeigen muß.
Val2	TimeValPtr, der auf den TimeVal-Record, der mit »Val1« verglichen werden soll, zeigen muß.
Ergebnis	INTEGER-Wert, der das Ergebnis des Vergleiches enthält. Ist »Ergebnis« größer als 0, wenn »Val1« größer ist als »Val2«, gleich 0, wenn beide gleich sind und kleiner als 0, wenn »Val1« kleiner ist als »Val2«.

Da wir soweit mit dem Timer-Device fertig wären, ist ein kleines Beispielprogramm sicher angebracht. Das folgende Demoprogramm mißt die Zeit, die das Programm für eine leere Schleife (1–100000) braucht:

```
MODULE TimerDemo;

FROM SYSTEM      IMPORT LONGSET,ADR;
FROM Exec        IMPORT MsgPortPtr,OpenDevice,DoIO,CloseDevice;

FROM ExecSupport IMPORT CreatePort,CreateExtIO,DeleteExtIO,DeletePort;
FROM Timer       IMPORT getSysTime,TimeRequest,TimeRequestPtr,TimeVal,vBlank,
                        SubTime;
FROM InOut       IMPORT WriteString,WriteLn,WriteCard;

VAR dport  : MsgPortPtr;
    dreq   : TimeRequestPtr;
    t1,t2  : TimeVal;
    i      : LONGCARD;

BEGIN
  dport:=CreatePort(NIL,Ø);           (* Port einrichten *)
  dreq:=CreateExtIO(dport,SIZE(TimeRequest));   (* Request einrichten *)

  (* Device öffnen *)
  OpenDevice(ADR("timer.device"),vBlank,dreq,LONGSET{}); (* Device öffnen *)

  dreq^.node.command:=getSysTime; (* Zeit vor der Berechnung holen *)
  DoIO(dreq);      (* ausführen! *)
  t1:=dreq^.time;     (* zwischenspeichern *)
```

```
  FOR i:=Ø TO 1ØØØØØØ DO END;    (* leere Schleife zwecks Zeitverschwendung *)

  dreq^.node.command:=getSysTime; (* Zeit nach der Berechnung holen *)
  DoIO(dreq);      (* ausführen! *)
  t2:=dreq^.time;    (* zwischenspeichern *)

  (* Ab hier kommt die Auswertung, die die Differenz der beiden Zeiten
     ausrechnet. Hier wurde der Einfachheit halber auf SUBTIME verzichtet. *)
  WriteLn; WriteString("Das Programm benötigte");
  WriteCard(t2.secs-t1.secs,1); WriteString(" Sekunden.");
  WriteLn;

  (* Arbeit beendet, hinterlasse geordneten Zustand! *)
  CloseDevice(dreq);     (* Device schließen *)
  DeleteExtIO(dreq);     (* Request vernichten *)
  DeletePort(dport);     (* Port vernichten *)
END TimerDemo.
```

8.5.6 Das Narrator-Device

Das Narrator-Device übernimmt die Sprachausgabe des Amiga. Leider übersetzt es nur Wörter oder Sätze, die in einer Art Lautschrift, den Phonem-Codes, vorhanden sind. Da die Umwandlung in diese Phonem-Codes teilweise recht umständlich ist, wurde die Translator-Bibliothek geschaffen, die aber nur TDI- und Benchmark-Benutzer öffnen müssen. Sie enthält nur einen einzigen Befehl für die Umwandlung von normaler Sprache in Phonem-Codes, doch dazu später.

Das Narrator-Device wird geöffnet wie die anderen (Name: »narrator.device«), doch auch hier unterscheidet sich der Request, den man benutzen sollte. Der für das Narrator-Device bestimmte Request heißt trefflicherweise Narrator und wird mit CREATEEXTIO initialisiert, sowie mit DELETEEXTIO gelöscht:

```
Narrator=RECORD
  message : IOStdReq ; normaler Request
  rate    : CARDINAL ; Sprechgeschwindigkeit in Wörtern pro Minute
  pitch   : CARDINAL ; Stimmlage
  mode    : CARDINAL ; Betonung
  sex     : CARDINAL ; Geschlecht
  chMasks : ADDRESS  ; benutzte Audio-Kanäle
  nmMasks : CARDINAL ; Anzahl der Kanäle
  volume  : CARDINAL ; Lautstärke
  sampFreq: CARDINAL ; Sampling-Frequenz
  mouths  : [Ø..255] ; Mundform
  chanMask: BYTE     ; welche Kanäle werden benutzt
  numChan : BYTE     ; Anzahl der Masken
  pad     : BYTE     ; fürs System
END;
```

rate In diesem Feld wird die Sprechgeschwindigkeit in Wörtern pro Minute angegeben. Der Defaultwert für dieses Feld ist 150, ich halte aber 120 für besser. Es gilt: je größer dieser Wert, desto schneller wird gesprochen. Werte von 40 bis 400 sind hier sinnvoll.

pitch Dieses Feld gibt die durchschnittliche Tonhöhe, in der gesprochen werden soll, an. Der Defaultwert hierfür ist 110. Die Tonhöhe sollte zwischen 65 (außerordentlich tief) und 320 (Gepiepse) variieren.

mode Hier können Sie angeben, ob der zu sprechende Text oszillisiert werden soll, was eine Art Betonung zur Folge hat. Möchten Sie, daß Ihr Text betont wird, so sollten Sie hier 0 (bzw. die Konstante natural aus Narrator) angeben. Soll der Text unbetont bleiben und sich somit »roboterhaft« anhören, müssen Sie in diesem Feld 1 (oder die Konstante robotic) eintragen. Geben Sie in diesem Feld nichts an, wird automatisch betont gesprochen.

sex Dieses Feld gibt das Geschlecht der Stimme, mit der gesprochen werden soll, an. 0 (oder die Konstante male) heißt, daß mit männlicher Stimme, 1 oder female mit weiblicher Stimme gesprochen wird. Der Defaultwert dieses Feldes ist 0.

chMasks Dieses Feld muß die Adresse eines Arrays mit den zu benutzenden Kanälen enthalten. Wie Sie vielleicht wissen, verfügt der Amiga über vier getrennt ansteuerbare Stimmen (sprich Kanäle). Diese haben die Nummern 3, 5, 10 und 12. In diesem Array können Sie nun angeben, welche der vier Kanäle Sie bei der Sprachausgabe ansteuern möchten.

nmMasks Hier müssen Sie die Anzahl der zu benutzenden Kanäle angeben.

volume In diesem Feld ist die Lautstärke, in der gesprochen werden soll, anzugeben. Der Defaultwert hierfür ist 64, die maximale Lautstärke.

sampFreq Dieses Feld muß die Sampling-Frequenz der Aussprache enthalten. Hier sind Werte von 5000 bis 28 000 möglich, wobei das Ergebnis um so klarer wird, je größer der Inhalt dieses Feldes ist. Für sampFreq beträgt der Defaultwert 22 200.

mouths Hier wird die Mundform angegeben, doch dazu kommen wir später.

Grundlegend für die vernünftige Sprachausgabe ist der TRANSLATE-Befehl der Translator-Bibliothek, der einen normalen String in Phonem-Codes übersetzt. Seine Syntax lautet:

ok := Translate (String , Stringlänge , Phonem , Phonemlänge);

String	ADDRESSe des Strings, der in Phonem-Codes übersetzt werden soll.
Stringlänge	LONGINT-Wert, der die Länge des zu übersetzenden Strings enthält.
Phonem	ADDRESSe eines Buffers, in dem die übersetzten Phonem-Codes als String abgelegt werden. Da ein Phonem-String immer länger ist als ein normaler, sollte dieser Buffer mindestens (je nach Stringlänge) 500–1000 Byte fassen können, bei manchen Anwendungen auch mehr.
Phonemlänge	LONGINT-Wert, der die Größe des Phonem-Buffers angibt.
ok	LONGINT-Wert, der entweder 0 (das Übersetzen verlief fehlerfrei) oder 1 (ein Fehler trat auf) enthält.

Haben Sie Ihren String fehlerfrei übersetzt, können Sie ihn mit dem Standardkommando WRITE an das Narrator-Device senden. Dabei müssen Sie natürlich auch die Spezialparameter wie Geschwindigkeit, Höhe, Kanäle, Lautstärke usw. dem Request übergeben:

command : write (oder #3)

data	Adresse des Buffers mit dem Phonem-String
length	Größe des Phonem-Buffers
rate	Sprechgeschwindigkeit (40–400)
pitch	Stimmlage (65–320)
mode	Betonung (natural/robotic)
sex	Geschlecht (male/female)
chMasks	Adresse des Kanal-Arrays
nmMasks	Anzahl der zu benutzenden Kanäle (1–4)
volume	Lautstärke (0–64)
sampFreq	Sampling-Frequenz (5000–28 000)

Rückgabewerte:

error	Evt. Fehlermeldung

Alles klar? Dann ist es Zeit für ein Demoprogramm. Experimentieren Sie ruhig mit den verschiedenen Werten!

```
MODULE NarratorDemo;

FROM SYSTEM       IMPORT LONGSET,ADR;
FROM Exec         IMPORT MsgPortPtr,OpenDevice,DoIO,CloseDevice,write,Byte;
FROM ExecSupport  IMPORT CreatePort,CreateExtIO,DeleteExtIO,DeletePort;
FROM Narrator     IMPORT Narrator,NarratorPtr,female,natural;
FROM Translator   IMPORT Translate;
FROM InOut        IMPORT ReadString,WriteString,WriteLn;
FROM Strings      IMPORT Length;

VAR dport  : MsgPortPtr;
    dreq   : NarratorPtr;
    String : ARRAY[Ø..6Ø] OF CHAR;
    Buffer : ARRAY[Ø..5ØØ] OF CHAR;
    Kanal  : ARRAY[Ø..3] OF Byte;
    error  : LONGINT;

BEGIN
  (* Ausgabekanäle definieren *)
  Kanal[Ø]:=3; Kanal[1]:=5; Kanal[2]:=1Ø; Kanal[3]:=12;

  dport:=CreatePort(NIL,Ø);           (* Port einrichten *)
  dreq:=CreateExtIO(dport,SIZE(Narrator));       (* Request einrichten *)

  OpenDevice(ADR("narrator.device"),Ø,dreq,LONGSET{}); (* Device öffnen *)

  WriteString("Zu sprechender Text:");           (* Text eingeben *)
  ReadString(String);
  WriteLn;

  (* Text übersetzen (in Phonem-Codes) *)
  error:=Translate(ADR(String),SIZE(String),ADR(Buffer),5ØØ);

  dreq^.message.command:=write;   (* Kommando: ausgeben *)
  dreq^.message.data:=ADR(Buffer);  (* Adresse des Phonem-Buffers *)
  dreq^.message.length:=5ØØ;   (* Länge *)
  dreq^.rate:=12Ø;     (* 1ØØ Wörter pro Minute *)
  dreq^.pitch:=23Ø;     (* Stimmlage 23Ø *)
  dreq^.sex:=female;     (* weibliche Stimme *)
  dreq^.mode:=natural;     (* betont *)
  dreq^.chMasks:=ADR(Kanal[Ø]);   (* Adresse des Kanal-Arrays *)
  dreq^.nmMasks:=4;     (* alle 4 Kanäle *)
  dreq^.volume:=64;     (* Lautstärke 64 *)
  dreq^.sampFreq:=222ØØ;   (* Samplingfrequenz 28ØØØ *)
  DoIO(dreq);     (* GO !! *)
```

```
  (* Arbeit beendet, hinterlasse geordneten Zustand ! *)
  CloseDevice(dreq);      (* Device schließen *)
  DeleteExtIO(dreq);      (* Request vernichten *)
  DeletePort(dport);      (* Port vernichten *)
END NarratorDemo.
```

8.5.6.1 Mundformen

Etwas ganz besonderes bietet das Narrator-Device mit den Mundformen, die sich synchron zum WRITE-Befehl mit READ abfragen lassen. Geben Sie nämlich 1 im Feld mouths des Narrator-Records an, sendet das Narrator-Device bei jeder Änderung der Form eines »Mundes« eine Rückmeldung. Voraussetzung für das Empfangen dieser Rückmeldung ist das Einrichten eines neuen Requests, des Mouth Records. Dieser wird mit CREATEEXTIO eingeführt. Da man ja parallel zum ersten Request arbeiten möchte, muß der Inhalt der Felder device und unit des Narrator-Records in die des Mouth-Records kopiert werden. Doch zunächst einmal die Form des Mouth-Records:

```
Mouth=RECORD
  voice : Narrator ; Narrator-Record
  width : [Ø..255] ; Breite des "Mundes"
  height: [Ø..255] ; Höhe des "Mundes"
  shape : BYTE     ; fürs System
  pad   : BYTE     ; fürs System
END;
```

Haben Sie diesen Record initialisiert, können Sie mit READ die aktuelle Mundform solange lesen, bis das entsprechende WRITE-Kommando ausgeführt und das letzte Wort des Phonem-Strings gesprochen wurde. Daß das Narrator-Device mit der Abarbeitung des letzten Wortes fertig ist, kann man daran erkennen, daß das READ-Kommando die Fehlermeldung NOWRITE (siehe Kapitel 8.5.6.2) zurückliefert. Hier der Inhalt der Felder des Request vor und nach dem DOIO-Aufruf:

command: read (oder #2)

Rückgabewerte:

width	Breite des »Mundes«
height	Höhe des »Mundes«
error	Evt. Fehlermeldung

Anhand der Breite und Höhe des »Mundes« kann dieser bequem auf dem Bildschirm nachgezeichnet werden.

8.5.6.2 Narrator-Fehlermeldungen

Das Narrator-Device kennt, bedingt durch die vielen Parameter, die übergeben werden müssen, einige Fehlermeldungen, die hier aufgeführt sind:

noMem	(#–2)	nicht genug Speicherplatz
noAudLib	(#–3)	Audio-Device kann nicht geöffnet werden
makeBad	(#–4)	Fehler im MAKELIBRARY-Aufruf
unitEr	(#–5)	anderes Unit als 0 wurde angegeben
cantAlloc	(#–6)	Audio-Kanäle nicht ansprechbar
unimpl	(#–7)	unimplementiertes Kommando
noWrite	(#–8)	WRITE-Befehl beim letzten Wort angelangt
expunged	(#–9)	Narrator-Device nicht ansprechbar
phonErr	(#–20)	Fehler im Phonem-String
rateErr	(#–21)	Sprechgeschwindigkeit außerhalb des Bereichs
pitchErr	(#–22)	Stimmlagenfehler
sexErr	(#–23)	anderes Geschlecht als 0 oder 1 gewählt
modeErr	(#–24)	ungültiger Modus (nicht 0 oder 1)
freqErr	(#–25)	ungültige Frequenz
volErr	(#–26)	ungültige Lautstärke

8.5.7 Das Audio-Device

Während es mit dem Narrator-Device nur möglich ist, Sprache auszugeben, kann man mit dem Audio-Device ganze Sounds erklingen lassen. Da die Soundfähigkeiten des Amiga schon von der einschlägigen Presse bejubelt wurden, sollen an dieser Stelle keine Worte mehr darüber verloren werden.

Das Audio-Device (»audio.device«) benötigt, wie Sie sich sicher schon gedacht haben, einen eigenen Request, den IOAudio-Request:

```
IOAudio=RECORD
  request : IORequest ; Standardrequest
  allocKey: INTEGER   ; Allozierungsschlüssel
  data    : ADDRESS   ; Adresse des Datenblocks
  length  : LONGCARD  ; Länge des Datenblocks
  period  : CARDINAL  ; Frequenz
  volume  : CARDINAL  ; Lautstärke
  cycles  : CARDINAL  ; Zyklen
  writeMsg: Message   ; Nachricht
END;
```

Um überhaupt etwas hören zu können, muß man sich einige der maximal vier Soundkanäle des Amiga bereitstellen. Das geschieht, indem Sie die Codes der Soundkanäle in einen Byte-Array legen (erinnert doch irgendwie an das Narrator-Device) und dessen

Adresse und Länge den Feldern data und length des IOAudio-Records (nicht des IOAudio^.request-Records!) vor dem OPENDEVICE-Befehl übergeben. Da der Amiga sogar stereotonfähig ist, können Sie sogar wählen, ob es sich bei dem bereitzustellenden Kanal um einen rechten oder linken handelt. Die Codes der Kanalkombinationen sind (die Zahlen kamen schon beim Narrator-Device vor, erinnern Sie sich?):

Kanal 3 links	Kanal 2 rechts	Kanal 1 rechts	Kanal Ø links	endgültiger Wert
Ø	Ø	1	1	3
Ø	1	Ø	1	5
1	Ø	1	Ø	1Ø
1	1	Ø	Ø	12

Somit entspricht 3 der ersten, 5 der zweiten, 10 der dritten und 12 der vierten Stimme.

Des weiteren müssen Sie vor dem OPENDEVICE-Aufruf dem Feld pri des IOAudio-Records (genauer gesagt dem Feld IOAudio^.request.message.node.pri) die Priorität Ihres Sounds übergeben. Je höher die Priorität, desto schwerer ist es für andere Programme, Ihnen die bereitgestellten Soundkanäle wegzunehmen, denn schließlich ist es in einem Multitasking-Betriebssystem möglich, mehrere Programme, die ebenfalls die Soundkanäle benutzen, laufenzulassen. Hat so ein Programm eine höhere Priorität als das Ihre, so werden Ihnen die Kanäle weggenommen. Folgende Prioritäten (»Precedences«) sind sinnvoll:

Precedence	Anmerkung
127	es ist für andere Programme unmöglich, Ihnen die bereitgestellten Kanäle wegzunehmen, da diese höchste Priorität haben
9Ø – 1ØØ	Sounds, die auf irgendwelche schwerwiegenden Fehler aufmerksam machen sollen
8Ø – 9Ø	Sounds, die den Benutzer auf etwas aufmerksam machen sollen
75	Sprachausgabe
5Ø – 7Ø	Sounds, die auf etwas noch nicht erkennbares hinweisen sollen
-5Ø – 5Ø	Musikstücke
-7Ø – Ø	Soundeffekte
-1ØØ – -8Ø	Hintergrundmusik
-127	niedrigste Priorität, jedes andere Programm kann Ihnen die Soundkanäle wegnehmen

Wenden wir uns nun der eigentlichen Soundprogrammierung zu. Dazu ist leider erst ein wenig Theorie nötig, aber Sie werden's schon schaffen.

Als erstes sollten wir uns mit den Wellenformen auseinandersetzen. Die Wellenform muß für das Audio-Device quasi als Wertetabelle der Kurve in einem Byte-Array abgelegt werden. Hierbei müssen Sie sich auf Werte von –127 bis 127 beschränken. Mit der Funktion SIN z.B. können sehr leicht solche Wertetabellen erstellt werden, in

diesem Fall für die Sinuskurve, die in gewissem Sinne »weich« klingt. Die Sägezahn-Wellenform, die einer linearen Funktion entspricht, klingt dagegen sehr »hart«. Beachten sollten Sie noch, daß jede Wellenform mit einem Nullbyte abschließen muß. Wenn Sie etwas Erfahrung mit der Programmierung von Synthesizern mitbringen, dürfte Ihnen das eigentlich bekannt sein. Hier gilt vor allen Dingen: ausprobieren!

Jetzt ist noch der Ton selbst festzulegen, was aber gar nicht so einfach ist. Für die Töne selbst existieren Frequenzen, die in der unteren Tabelle aufgeführt sind. Um nun die »Sampling Period«, die das Audio-Device benötigt, zu errechnen, muß man folgende Formel anwenden:

```
                        1
Period = ------------------------------
         Frequenz * 32 * 2.79365 * 10^-7
```

(Die 32 im Nenner ist kein konstanter Wert, sie bezieht sich auf Byteanzahl von 32 pro Ton, was aber im Normalfall genügt. Unter Umständen müssen Sie hier Ihre eigene Anzahl eintragen.)

Hier sind die Frequenzen der Töne, die produziert werden können (die Zahl hinter der Note gibt jeweils die Oktave an, das #-Zeichen bedeutet eine Erhöhung um einen halben Ton):

Bemerkung	Note	Frequenz
	A0	27.50000
	A#0	29.13523
	H0	30.86770
	C1	32.70319
	C#1	34.64782
	D1	36.70809
	D#1	38.89087
	E1	41.20344
	F1	43.65352
	F#1	46.23930
	G1	48.99942
	G#1	51.91308
	A1	55.00000
	A#1	58.27047
	H1	61.73543
	C2	65.40639
	C#2	69.29565
	D2	73.41619
	D#2	77.78174
	E2	82.40688

	F2	87.30705
Baß	F#2	92.49860
	G2	97.99885
	G#2	103.82617
	A2	110.00000
	A#2	116.54094
Bariton	H2	123.47082
	C3	130.81278
Tenor	C#3	138.59131
	D3	146.83238
	D#3	155.56349
	E3	164.81377
	F3	174.61411
Alt	F#3	184.99721
	G3	195.99771
	G#3	207.65234
	A3	220.00000
Mezzosopran	A#3	233.08188
	H3	246.94165
	C4	261.62556
Sopran	C#4	277.18263
	D4	293.66476
	D#4	311.12698
	E4	329.62755
	F4	349.22823
	F#4	369.99442
	G4	391.99543
	G#4	415.30469
Kammerton	A4	440.00000
	A#4	466.16376
	H4	493.88330
	C5	523.25113
	C#5	554.36526
	D5	587.32953
	D#5	622.25396
	E5	659.25511
	F5	698.45646
	F#5	739.98884
	G5	783.99087
	G#5	830.60939
	A5	880.00000
	A#5	932.32752
	H5	987.76660

	C6	1046.50226
	C#6	1108.73052
	D6	1174.65907
	D#6	1244.50793
	E6	1318.51022
	F6	1396.91292
	F#6	1479.97769
	G6	1567.98174
	G#6	1661.21879
	A6	1760.00000
	A#6	1864.65504
	H6	1975.53320
	C7	2093.00452
	C#7	2217.46104
	D7	2349.31814
	D#7	2489.01586
	E7	2637.02045
	F7	2793.82585
	F#7	2959.95538
	G7	3135.96348
	G#7	3322.43758
	A7	3520.00000
	A#7	3729.31009
	H7	3951.06641
	C8	4186.00904
	C#8	4434.92209
	D8	4698.63628
	D#8	4978.03173
	E8	5274.04091
	F8	5587.65170
	F#8	5919.91076
	G8	6271.92697
	G#8	6644.87516
Aliasing Distortion	A8	7040.00000
	A#8	7458.62018
	H8	7902.13282
	C9	8372.01808
	C#9	8869.84419
	D9	9397.27257
	D#9	9956.06347
	E9	10548.08182
	F9	11175.30340
	F#9	11839.82152

```
G9      12543.85395
G#9     13289.75Ø32
A9      14Ø8Ø.ØØØØØ
A#9     14917.24Ø36
```

»Aliasing Distortion« heißt ein technisches Phänomen, bei dem ab einer bestimmten Frequenz Störfrequenzen hörbar sind. Doch das kommt nur in Tonbereichen vor, die man nur sehr selten braucht, nämlich ab A8.

Nun können Sie Ihre Sounds mit WRITE abspielen:

command : write (oder #3)

flags	Hier müssen die IOFlags 0 und 4 (IOFlagSet{0,4}) gesetzt sein, damit das Audio-Device weiß, daß eine Lautstärkeangabe kommt und daß die ankommenden Daten sofort der Hardware mitgeteilt werden sollen (diese Flags heißen QUICK und PERVOL).
data	(im IOAudio-Record) Adresse des Byte-Arrays mit der Wellenform.
length	(im IOAudio-Record) Größe des Byte-Arrays, der die Wellenform enthält.
period	»Sampling Period« des zu spielenden Tones.
cycles	Länge des zu spielenden Tones. Je größer dieser Wert ist, desto länger wird der Ton angehalten.
volume	Lautstärke des Tones.
Rückgabewerte:	
error	Evt. Fehlermeldung

Dieser Request wird dem Audio-Device nicht etwa mit DOIO, wie man vermutet, geschickt, sondern mit BEGINIO aus ExecSupport:

BeginIO (Request);

Request	ADDRESSe des Requests.

Sie können Ihr Programm nun warten lassen, bis das Audio-Device mit der Ausgabe des Tons fertig ist. Dazu dient der WAITIO-Befehl aus Exec:

WaitIO (Request);

Request	ADDRESSe des Requests.

Jetzt ist es Zeit für ein Demoprogramm:

```
MODULE AudioDemo;

FROM SYSTEM      IMPORT LONGSET,ADR;
FROM Exec        IMPORT MsgPortPtr,OpenDevice,CloseDevice,write,Byte,WaitIO,
                        IOFlagSet;
FROM ExecSupport IMPORT CreatePort,CreateExtIO,DeleteExtIO,DeletePort,BeginIO;
FROM Audio       IMPORT IOAudio,IOAudioPtr;

VAR dport  : MsgPortPtr;
    dreq   : IOAudioPtr;
    Kanal  : ARRAY[Ø..3] OF Byte;
    Sound  : ARRAY[Ø..127] OF Byte;
    error  : LONGINT;
    i      : INTEGER;

BEGIN
  (* Ausgabekanäle definieren *)
  Kanal[Ø]:=3; Kanal[1]:=5; Kanal[2]:=1Ø; Kanal[3]:=12;

  dport:=CreatePort(NIL,Ø);          (* Port einrichten *)
  dreq:=CreateExtIO(dport,SIZE(IOAudio));         (* Request einrichten *)
  dreq^.request.message.node.pri:=4Ø;  (* Priorität *)
  dreq^.data:=ADR(Kanal[Ø]);   (* Soundkanäle *)
  dreq^.length:=4;         (* Anzahl der Kanäle *)
  OpenDevice(ADR("audio.device"),Ø,dreq,LONGSET{}); (* Device öffnen *)

  FOR i:=Ø TO 126 DO     (* Wellenform initialisieren *)
    Sound[i]:=i;     (* Sägezahn (linear) *)
  END;
  Sound[127]:=Ø;     (* Abschlußbyte *)

  dreq^.request.command:=write;    (* Sound anhören *)
  dreq^.request.flags:=IOFlagSet{Ø,4};  (* Flags QUICK und PERVOL *)
  dreq^.data:=ADR(Sound[Ø]);   (* Adresse der Wellenform *)
  dreq^.length:=127;      (* Länge der Wellenform *)
  dreq^.period:=1Ø7;      (* Ton, hier C6 *)
  dreq^.volume:=64;      (* Lautstärke *)
  dreq^.cycles:=5ØØ;      (* Dauer *)
  BeginIO(dreq);      (* GO !!! *)
  WaitIO(dreq);       (* Warten ... *)
```

```
  (* Arbeit beendet, hinterlasse geordneten Zustand! *)
  CloseDevice(dreq);      (* Device schließen *)
  DeleteExtIO(dreq);      (* Request vernichten *)
  DeletePort(dport);      (* Port vernichten *)
END AudioDemo.
```

8.5.7.1 Weitere Audio-Device-Befehle

Um die Priorität eines Kanals zu ändern, sollten Sie den SETPREC-Befehle verwenden:

command : setPrec

pri	Neue Priorität
Rückgabewerte:	
error	Evt. Fehlermeldung

(Dieser Befehl wird übrigens mit DOIO ausgeführt.)

Es existieren auch zwei Befehle, mit denen man die Soundausgabe unterbrechen und wieder fortsetzen kann, STOP und START. Zuerst STOP, der die Soundausgabe unterbricht:

command : stop

Rückgabewerte:	
error	Evt. Fehlermeldung

Und nun START, der sie wieder fortsetzt:

command : start

Rückgabewerte:	
error	Evt. Fehlermeldung

8.5.7.2 Programmierung mehrerer Stimmen

Die Ausgabe mehrstimmiger Sounds (oder gar Musikstücke) ist im Prinzip so wie die einstimmiger Sounds, mit dem Unterschied, da bei mehreren Stimmen für jede Stimme ein IOAudio-Request generiert werden muß. Jede Stimme wird getrennt angesteuert. Auf ein Beispielprogramm wurde an dieser Stelle bewußt verzichtet, da es von Ihnen ohne große Mühen selbst programmiert werden kann.

8.5.7.3 Audio-Device-Fehlermeldungen

Die Fehlermeldungen, die man eventuell im Feld error nach einem DOIO- oder BEGINIO-Befehl zurückbekommen könnte, sind im Audio-Modul definiert:

noAllocation (#–10) Der Schlüssel für die Bereitstellung eines Soundkanals stimmt nicht.

allocFailed (#–11) Der betreffende Soundkanal konnte nicht bereitgestellt werden, da z.B. ein anderes Programm mit höherer Precedence diesen für seine Zwecke beansprucht.

channelStolen (#–12) Der betreffende Soundkanal wurde von einem Programm mit höherer Precedence für seine Zwecke »beschlagnahmt«.

8.5.8 Das Clipboard-Device

Das Clipboard-Device dient der Datenübermittlung zwischen den einzelnen Tasks. Wenn z.B. eine Textverarbeitung und eine parallel laufende Dateiverwaltung das Clipboard-Device unterstützen, ist ein außerordentlich einfacher Datenaustausch möglich. Doch auch ein Programm allein kann das Clipboard-Device sehr effektiv unterstützen. Wenn Sie viel mit mehr oder weniger großen Datenblöcken arbeiten, können Sie diese einfach auf dem Clipboard-Device ablegen. Sie werden dort solange festgehalten, bis Sie sie überschreiben oder löschen. Paßt ein Datenblock nicht auf das Clipboard-Device, wird er im Verzeichnis SYS:devs/clipboards gespeichert. Sollte das Verzeichnis devs/clipboards nicht auf der Startdiskette vorhanden sein, legt das System automatisch ein solches an, vorausgesetzt, die Diskette ist nicht schreibgeschützt.

Für das Clipboard-Device ist ein spezieller Request nötig:

```
IOClipReq=RECORD
  message: Message      ; Nachricht
  device : DevicePtr    ; Zeiger auf das Device
  unit   : UnitPtr      ; Zeiger auf die Einheit
  command: CARDINAL     ; Kommando
  flags  : IOFlagSet    ; Flags
  error  : [-128..127]  ; Fehlermeldung
  actual : LONGCARD     ; übertragene Bytes
  length : LONGCARD     ; Länge eines Datenblocks
  data   : ADDRESS      ; Datenblock
  offset : LONGCARD     ; Offset in einem Datenblock
  clipID : LONGINT      ; Identifizierungsnummer eines Datensatzes
END;
```

ClipID gibt übrigens auch an, wie oft ein Datenblock geschrieben und wieder gelöscht wurde.

Das Clipboard-Device ist mit einer (sehr) kleinen Dateiverwaltung vergleichbar. Eine Einheit entspricht einem Datenblock auf dem Clipboard-Device. Jede Einheit hat ihre eigene Identifikationsnummer (ID) und jedes Byte einer Einheit ist durch Offsets ansteuerbar.

Das Clipboard-Device wird wie üblich mit OPENDEVICE (Name: »clipboard.device«) geöffnet. Sie müssen allerdings eine Unit-Nummer angeben. Diese Unit-Nummer gibt den Datensatz des Devices, den Sie bearbeiten möchten, an. Beim ersten Schreib-/Lesezugriff muß dieser selbstverständlich 0 sein.

Kommen wir zum Schreiben von Daten auf das Clipboard-Device. Hierfür wird der Befehl WRITE verwendet:

command : write (oder #3)

clipID	Beim ersten Schreibzugriff 0, sonst nicht modifizieren!
data	Adresse des zu schreibenden Datenblocks
length	Länge des zu schreibenden Datenblocks
offset	Byte-Offset, ab dem zu schreiben ist. Beim ersten Schreibzugriff muß dieses Feld natürlich 0 enthalten. Sollte Ihnen der Begriff Offset noch nicht ganz klar sein, ist es angebracht, mal im DOS- oder TrackDisk-Kapitel nachzusehen.

Rückgabewerte:

actual	Anzahl der tatsächlich geschriebenen Bytes.
error	Evt. Fehlermeldung

Um dem Clipboard-Device klarzumachen, daß der geschriebene Block komplett ist und auch wieder lesbar sein soll, muß nach dem WRITE-Kommando ein UPDATE-Kommando folgen:

command : update (oder #4)

Rückgabewerte:

error	Evt. Fehlermeldung

Nun wäre es unsinnig, Daten abzulegen, die man nicht wieder holen kann. Aber da das Clipboard-Device keine unsinnige Einrichtung ist, kann man mit dem READ-Befehl einen Block wieder lesen:

command : read (oder #2)

clipID	Beim ersten Lesezugriff 0, sonst aber nicht modifizieren!
data	Adresse des Blocks, in den die Daten eingelesen werden sollen.
length	Größe des Einleseblocks.
offset	Byte-Offset, ab dem zu lesen ist. Beim ersten Lesezugriff sollte dieser natürlich 0 sein.

Rückgabewerte:

actual	Hier sollte eigentlich die Anzahl der tatsächlich gelesenen Bytes stehen, doch fälschlicherweise wird hier die Größe des Einleseblocks hineingeschrieben.
error	Evt. Fehlermeldung

Benötigen Sie einen Block nicht mehr, ist es ratsam, ihn mit CLEAR zu löschen. Dabei wird der clipID-Zähler um eins erhöht:

command : clear (oder #5)

Rückgabewerte:

error	Evt. Fehlermeldung

Hier ein Beispielprogramm:

```
MODULE ClipDemo;

FROM SYSTEM      IMPORT LONGSET,ADR;
FROM Exec        IMPORT MsgPortPtr,OpenDevice,DoIO,CloseDevice,write,read,
                        update,clear;
FROM ExecSupport IMPORT CreatePort,CreateExtIO,DeleteExtIO,DeletePort;
FROM ClipBoard   IMPORT IOClipReq;
FROM InOut       IMPORT WriteLn,WriteString;

TYPE IOClipReqPtr=POINTER TO IOClipReq;

VAR dport  : MsgPortPtr;
    dreq   : IOClipReqPtr;
    String : ARRAY[Ø..5Ø] OF CHAR;
    Daten  : ARRAY[Ø..5Ø] OF CHAR;
```

```
BEGIN
  String:="Hallo! Dies ist ein ClipBoard-Text!!";

  dport:=CreatePort(NIL,Ø);          (* Port einrichten *)
  dreq:=CreateExtIO(dport,SIZE(IOClipReq));      (* Request einrichten *)

  (* Device öffnen *)
  OpenDevice(ADR("clipboard.device"),Ø,dreq,LONGSET{}); (* Device öffnen *)

  dreq^.command:=write;     (* WRITE für schreiben *)
  dreq^.clipID:=Ø;     (* erster Datensatz *)
  dreq^.data:=ADR(String);    (* String schreiben *)
  dreq^.length:=SIZE(String);    (* Stringlänge *)
  dreq^.offset:=Ø;     (* Offset Ø (1.Mal) *)
  DoIO(dreq);      (* GO!! *)

  dreq^.command:=update;    (* UPDATE senden *)
  DoIO(dreq);      (* ausführen! *)

  dreq^.command:=read;     (* READ, um zu lesen *)
  dreq^.clipID:=Ø;     (* erster Datensatz *)
  dreq^.data:=ADR(Daten);    (* in "Daten" lesen *)
  dreq^.length:=SIZE(Daten);    (* maximal soviel lesen *)
  dreq^.offset:=Ø;     (* Offset Ø *)
  DoIO(dreq);      (* GO!! *)

  dreq^.command:=clear;     (* CLEAR, um zu löschen *)
  DoIO(dreq);      (* GO!! *)

  WriteString(Daten); WriteLn;    (* Daten ausgeben! *)

  (* Arbeit beendet, hinterlasse geordneten Zustand! *)
  CloseDevice(dreq);     (* Device schließen *)
  DeleteExtIO(dreq);     (* Request vernichten *)
  DeletePort(dport);     (* Port vernichten *)
END ClipDemo.
```

8.5.8.1 Anmelden von Schreib-/Lesezugriffen

Möchten Sie einen Datenblock dem Clipboard-Device zur Verfügung stellen, aber doch erfahren, ob ein Programm versucht, diesen zu lesen, melden Sie Ihren Block doch einfach an. Dazu müssen Sie mit CREATEPORT einen neuen Nachrichtenport initialisieren und diesen dem POST-Befehl übergeben:

command : post	
data	Adresse des neuen Ports
clipID	Nummer des Datentranfers
Rückgabewerte:	
clipID	Neue Datentransfernummer
error	Evt. Fehlermeldung

Sie können nun, wie bei den IDCMP-Nachrichten, mit GETMSG abfragen, ob eine Nachricht in Ihrem Spezialport eingetroffen ist. Ist eine eingetroffen, hat sie folgende Form:

```
SatisfyMsg=RECORD
  msg   : Message  ; normale Nachricht
  unit  : CARDINAL ; Nummer des Datensatzes
  clipID: LONGINT  ; Nummer des Datentransfers
END;
```

Haben Sie so eine Nachricht erhalten, sollte Sie dem Clipboard-Device mit WRITE Ihren Datenblock schleunigst zur Verfügung stellen, schließlich wartet ein anderes Programm darauf. Für diese Zwecke gibt es noch zwei weitere Befehle: CURRENT-READID und CURRENTWRITEID. Diese dienen dazu, die clipID-Werte des Requests des gerade vom Device lesenden oder des zuletzt auf das Device schreibenden Programms zu holen. Sie können sie dann mit den Werten Ihres Requests vergleichen und so Rückschlüsse ziehen, doch dazu gleich. Hier erst die Syntax der beiden Befehle:

command : currentReadId	
Rückgabewerte:	
clipID	ClipID-Wert des gerade vom Device lesenden Programms

command : currentWriteId	
Rückgabewerte:	
clipID	ClipID-Wert des zuletzt auf das Device schreibenden Programms

Anhand der zurückgelieferten clipID kann man sehen, ob das andere Programm Ihre Daten, die Sie zur Verfügung gestellt haben, benötigt. Die SatisfyMsg bekommt man

nämlich immer, wenn auf das Clipboard-Device zugegriffen wird, dabei müssen es nicht Ihre Daten sein, die gewünscht werden. Ist die neue clipID nämlich größer als die alte, werden Ihre Daten nicht benötigt.

8.5.8.2 Clipboard-Fehlermeldungen

Das Clipboard-Device kennt nur eine Fehlermeldung, die es zurückliefern kann, OBSOLETEID (#1). OBSOLETEID heißt schlicht und einfach, daß die ClipID-Nummer veraltert ist.

8.6 Anmerkungen und zusätzliche Befehle

Das erste, was in diesem Unterkapitel besprochen werden soll, ist das Öffnen von Bibliotheken, was den Benchmark- und TDI-Benutzern ihr Compiler nicht abnimmt. Eine Bibliothek wird mit OPENLIBRARY aus der Exec-Library, die stets offen ist, geöffnet:

Bibliothek := OpenLibrary (Name , Version);

Name	ADDRESSe des Strings mit dem Bibliotheksnamen.
Version	LONGINT-Wert, der die Bibliotheksversion angibt. Bei Kickstart v1.2 muß dieser Parameter 33 sein.
Bibliothek	LibraryPtr auf die soeben geöffnete Bibliothek. Je nach Library ist dies ein GfxBasePtr (GRAPHICS), ein IntuitionBasePtr (Intuition), ein DosBasePtr (DOS), ...

Am Programmende muß eine Library wieder mit CLOSELIBRARY geschlossen werden:

CloseLibrary (Bibliothek);

Bibliothek	LibraryPtr auf die zu schließende Library.

Nun sollte noch gesagt werden, daß wir in diesem Kapitel nicht die ganzen Funktionen von EXEC besprechen konnten, das ist vielleicht die Aufgabe eines ganzen Buches, nicht aber eines Kapitels. Doch Sie wissen nun, was es mit den wichtigsten Dingen von EXEC auf sich hat, den Tasks und vor allem den bedeutsamsten Devices, und das soll auch für dieses Buch reichen. Im Literaturanhang finden Sie Bücher, die sich zum größten Teil mit EXEC befassen, doch leider beziehen sich diese auf C- und Maschinensprache.

Anhang A

Die Guru-Meditationen

Ja, wer kennt sie nicht – die seltsamen Nummern im rotblinkenden Kasten nach einem Programmabsturz. Entgegen der landläufigen Meinung, die Zahlencodes seien wenig aussagend oder gar schwachsinnig, muß ich sagen, daß sie eine tolle Eigenschaft des AMIGA sind, da man bei richtiger Deutung wenigstens einen Tip bekommt, wo der Fehler liegen könnte. Andere Computer verabschieden sich dagegen sang- und klanglos und erschweren somit die Fehlersuche ungemein.

Zur (nicht uninteressanten) Geschichte der Guru-Meditationen sei gesagt, daß sie von einem Joyboard (einem Skateboard mit Joystickfunktionen, um es grob zu beschreiben) abstammen, denn sobald ein Programmierfehler auftrat, setzte sich der entsprechende Programmierer der Amiga Inc., (Die Firma, die Joysticks herstellte, um die Entwicklung des AMIGAs zu finanzieren, und dann später, eben des Geldes wegen, von dem Commodore etwas mehr hatte, von Commodore gekauft wurde.) auf dieses wacklige Board und versuchte, sich solange auf ein Problem zu konzentrieren, bis das Joyboard völlig ruhig war (eine nicht zu unterschätzende Leistung, wenn man weiß, wie wacklig ein Skateboard ist). R. J. Mical, der den Großteil der Intuition Software schrieb, hatte vor, ein Spiel zu programmieren, bei dem es das Ziel war, ins Nirwana zu kommen (der höchste Wunsch aller Buddhisten), indem man das Joyboard möglichst schnell in den Ruhezustand versetzte, aber irgendwie war er wohl zu beschäftigt, neben der Entwicklung des AMIGA noch ein Game zu schreiben. Wie dem auch sei, da man in Systemabstürzen Gründe zum Meditieren sah, wurden die Systemfehlermeldungen »Guru-Meditations« genannt.

Wenden wir uns ernsteren Dingen zu. Wie sind die Guru-Meditationen aufgebaut?

AA BB CC DDDD. EEEEEEEE

EEEEEEEE	Hier steht die Adresse, an der der Fehler auftrat.
AA	Fehlertyp, entweder 00 (RECOVERY-ALERT, das System kann durch Drücken der linken Maustaste wieder in den Normalzustand versetzt werden. Einen RECOVERY-ALERT erkennt man auch daran, daß er das Display nach unten schiebt) oder 80 (DEAD-

END-ALERT, der Fehler ist so gravierend, daß ein Reset ausgelöst werden muß, um weiterarbeiten zu können. DEADEND-ALERTs lassen den Bildschirm komplett schwarz werden).

BB Betriebssystemteil (Subsystem), in dem der Fehler auftrat. Folgende Codes sind bekannt:

68000er CPU

00: TRAP (Prozessortrap)

Bibliotheken

01: EXEC (EXEC-Bibliothek)
02: GRAPHICS (GRAPHICS-Bibliothek)
03: LAYERS (LAYERS-Bibliothek)
04: INTUITION (INTUITION-Bibliothek)
05: MATH (MATH-Bibliothek)
06: CLIST (CLIST-Bibliothek)
07: DOS (DOS-Bibliothek)
08: RAM (RAM-Bibliothek)
09: ICON (ICON-Bibliothek)
0A: EXPANSION (EXPANSION-Bibliothek)

Devices

10: AUDIO (AUDIO-Device)
11: CONSOLE (CONSOLE-Device)
12: GAMEPORT (GAMEPORT-Device)
13: KEYBOARD (KEYBOARD-Device)
14: TRACKDISK (TRACKDISK-Device)
15: TIMER (TIMER-Device)

Resources

20: CIA (Prozessor)
21: DISK (Diskettenhardware)
22: MISC (Verschiedenes)

Sonstiges

30: BOOTSTRAP (Bootroutine)
31: WORKBENCH (Workbenchfehler)
32: DISKCOPY (DiskCopy-Fehler)

Ist AA 0 (Prozessortrap), so handelt es sich nicht um einen Software- sondern um einen Hardwarefehler, der durch den 68000er Chip bedingt ist (und sich abfangen läßt, wie Sie wissen). Es gibt folgende Traps:

Traps (komplette Fehlermeldungen BB CC DDDD)

00000002	Timingfehler auf dem Adreß- oder Datenbus
00000003	Adressierungsfehler (sehr, sehr häufig!)
00000004	Illegale Instruktion (dieser Guru tritt bei mir am häufigsten auf!)
00000005	Division durch 0
00000006	CHK-Fehler, Grenzen eines Zahlenbereichs überschritten
00000007	TRAPV-Instruktion, absichtlicher Trapfehler
00000008	Privilegverletzung
00000009	Einzelschrittmodus
0000000A	Line-A-Emulator (Illegaler Opcode 1010)
0000000B	Line-F-Emulator (Illegaler Opcode 1111)
CC	Diese beiden Buchstaben bestimmen einen übergeordneten Fehler:
00	Fehler kann nicht zugeordnet werden
01	Speicherplatzmangel
02	Library konnte nicht generiert werden
03	Library konnte nicht geöffnet werden
04	Device konnte nicht geöffnet werden
05	Hardware-Baustein reagiert nicht (Achtung!!)
06	Ein-/Ausgabefehler
07	Kein Signal

Für jeden Subsystem-Fehler existieren noch einige spezielle Fehlermeldungen, Format BBCCDDDD:

EXEC-Bibliothek

01000001	CPU-Prüfsummenfehler bei Exceptions (Ausnahmezuständen des Prozessors)
81000002	Prüfsummenfehler bei Berechnung der ExecBase-Startadresse

81000003	Prüfsummenfehler bei einer Library (wahrscheinlich ist die Library zerstört)
81000004	Nicht genug Speicher für eine Library
81000005	Zerstörte Speicherverwaltungsliste
81000006	Nicht genug Speicher für Interruptbehandlung
81000007	Zeigerfehler
81000008	Signal (Semaphore) zerstört
81000009	Speicherbereich wurde zum zweiten Mal freigegeben (recht häufig)
8100000A	Es wurden noch nicht belegte Vektoren für spätere Erweiterungen belegt

GRAPHICS-Bibliothek

82010001	Nicht genug Speicherplatz für eine Copperliste
82010002	Nicht genug Speicherplatz für eine Copper-Instruktionsliste
82000003	Copperliste ist voll (mehr als 10000 Einträge)
82000004	Struktur der Copperliste ist zerstört
82010005	Nicht genug Speicherplatz für den Kopf der Copperliste
82010006	Nicht genug Speicherplatz für die erste Interlace-Copperliste
82010007	Nicht genug Speicherplatz für die zweite Interlace-Copperliste
82010008	Nicht genug Speicherplatz für AREAs oder FLOOD (kein temporäres Raster reserviert?)
82010009	Nicht genug Speicherplatz für ein temporäres Raster
8201000A	Nicht genug Speicherplatz für die Blitter-Bitmap
8201000B	Falscher Speicherbereich
82010030	Fehler beim Einrichten eines ViewPorts
82011234	Kein Zwischenspeicher

LAYERS-Bibliothek

83010001	Nicht genug Speicherplatz für ein Layer

INTUITION-Bibliothek

84000001	Unbekannter Gadgettyp
84010002	Nicht genug Speicherplatz für die Erstellung eines Ports
84010003	Nicht genug Speicherplatz für eine Menüleiste und/oder eines Menüpunktes
84010004	Nicht genug Speicherplatz für einen Untermenüpunkt (Subitem)
84010005	Nicht genug Speicherplatz für einen Menüoberbegriff
84000006	Falsche Position der Menüleiste
84010007	Nicht genug Speicherplatz für einen Screen
84010008	Nicht genug Speicherplatz für einen RastPort
84000009	Unbekannter Screentyp
8401000A	Nicht genug Speicherplatz für ein Gadget
8401000B	Nicht genug Speicherplatz für ein Window
8400000C	Statusregister hat falschen Inhalt beim Öffnen der Intuition-Bibliothek
8400000D	Falsche IDCMP-Nachricht
8400000E	Nicht genug Speicherplatz für den Nachrichtenstack
8400000F	Das Console-Device kann nicht geöffnet werden

DOS-Bibliothek

07010001	Nicht genug Speicherplatz beim StartUp
07000002	Der ENDTASK-Befehl wirkte fehlerhaft oder gar nicht, was zur Folge hat, daß der Task nicht beendet wurde
07000003	Fehler beim Übertragen eines Datenpaketes
07000004	Datenpaket empfangen, das nicht erwartet wurde
07000005	Der FREEVEC-Befehl wirkte fehlerhaft oder gar nicht, was zur Folge hat, daß ein Zeiger nicht befreit wurde
07000006	Fehlerhafte Daten in einem Diskettenblock
07000007	Fehlerhafte oder zerstörte Bitmap
07000008	Filenummer bereits gelöscht

07000009	Fehlerhafte Prüfsumme eines Diskettenblocks (z.B. Rootblock)
0700000A	Diskettenfehler
0700000	Filenummer außerhalb der zulässigen Bereichs
0700000C	Der »Overlay-Hunk« in der Bootsequenz ist fehlerhaft

RAM-Bibliothek

08000001	Fehlerhafter Eintrag in der Speicherverwaltungsliste

EXPANSION-Bibliothek

0A000001	Hard- oder Softwarefehler bei einer Hardware-Erweiterung

TRACKDISK-Device

14000001	Fehler während des Suchens auf Diskette
14000002	Fehler bei einem Timper-Impuls

TIMER-Device

15000001	Fehler beim Zugriff (bzw. Zugriffsversuch) auf das Timer-Device
15000002	Netzfrequenz instabil: Fehler bei der Zeitkoordinierung

DISK-Resource

21000001	Ungültiges DISKCHANGE-Signal: eingelegte Diskette wurde nicht erkannt
21000002	Kein aktives Laufwerk vorhanden

BOOTSTRAP

30000001	DOS-Bibliothek nicht gefunden oder Fehler beim Auswerten der Bootblock-Daten (kann passieren, wenn man zu sehr am Bootblock herumfummelt und versucht, da irgendwelche fehlerhaften Routinen hineinzuzwängen)

Es ist übrigens möglich, daß mehrere Fehler zugleich auftreten, wenn auch selten. In diesem Fall werden einfach die verschiedenen Codes addiert.

Ich hoffe, dieser Anhang trägt dazu bei, daß Sie Ihre Fehler schneller erkennen oder zumindest wissen, was es mit den geheimnisvollen Nummern auf sich hat.

Anhang B

Tips zur Umsetzung von Programmen anderer Sprachen

Da in den meisten Büchern und Zeitschriften die Sprache C benutzt wird, möchte ich hier einige Tips bringen, wie man C-Programme umsetzt. Da auch die Sprache Basic nicht selten verwendet wird, möchte ich in diesem Anhang auch Hinweise geben, wie man Basic-Programme umsetzen kann.

Zunächst aber zur Umsetzung von C-Programmen. Sie haben im Laufe des Buches gesehen, daß bei der Programmierung des Betriebssystems viele Daten in Records übergeben werden. Auch in C gibt es eine Art Records, die Structs:

```
struct Struktur {
   UWORD Eintrag1;
   Struktur *Eintrag2;
};
struct Struktur *MeineStruktur
```

In MODULA-2 wird diese Struktur folgendermaßen definiert:

```
TYPE StrukturPtr=POINTER TO Struktur
     Struktur=RECORD
       Eintrag1: CARDINAL;
       Eintrag2: StrukturPtr;
     END;
VAR MeineStruktur: StrukturPtr;
```

Das sagt schon einiges über die Handhabung von Zeigern in C aus. Das Feld eines Zeigers wird folgendermaßen angesprochen:

```
MeineStruktur->Eintrag1 = (UWORD)15;
```

```
(MODULA-2: MeineStruktur^Eintrag1:=15; )
```

Ein normales Record-Feld hingegen wird so gehandhabt (angenommen »MeineStruktur« sei nun kein Zeiger, sondern ein Record-Typ):

```
MeineStruktur.Eintrag1 = (UWORD)15;
```

```
(MODULA-2: MeineStruktur.Eintrag1:=15; )
```

Ähnlich den Structs sind die Unions, die Verbundtypen. In einer Union sind mehrere Typen definiert, von denen aber immer nur einer zutrifft. Eine typische C-Union sieht so aus:

```
struct Struktur
{
   union Eintrag
   {
     UWORD a;
     UWORD b;
   };
   Struktur *Eintrag2;
};
struct Struktur *MeineStruktur;
```

In MODULA-2 kann dieses Problem mit einer CASE-Anweisung sehr elegant gelöst werden:

```
TYPE StrukturPtr=POINTER TO Struktur;
     Struktur=RECORD
       CASE :CARDINAL OF
       |0: a: CARDINAL;
       |1: b: CARDINAL;
       END;
     END;
VAR MeineStruktur: StrukturPtr;
```

Zugegriffen wird auf Unions bzw. unsere »Pseudo-Unions« wie auf einfache Structs bzw. Records.

Ein zweiter, sehr wichtiger Faktor sind die Datentypen. In C wird eigentlich viel Wirbel um wenige grundverschiedene Datentypen gemacht. Hier eine Konversionstabelle (die meisten C-Datentypen sind im C-Includefile <exec/types.h> definiert):

C	MODULA-2
char	CHAR
unsigned char	CHAR
short	INTEGER
unsigned short	CARDINAL
long	LONGINT
unsigned long	LONGCARD
float	REAL oder FFP
double	REAL

BYTE	Byte
UBYTE	UByte
BYTEBITS	SET OF [0..7]
WORD	INTEGER
UWORD	CARDINAL
WORDBITS	BITSET
LONG	LONGINT (selten auch ADDRESS)
ULONG	LONGCARD
LONGBITS	LONGSET
STRPTR	ADDRESS oder POINTER TO CHAR
APTR	ADDRESS
CPTR	ADDRESS
SHORT	INTEGER
USHORT	CARDINAL
FLOAT	REAL oder FFP
DOUBLE	REAL
COUNT	INTEGER
UCOUNT	CARDINAL
BOOL	BOOLEAN (eigentlich INTEGER)
TEXT	CHAR

Bei den vielen Zahlenwerten, die beide Sprachen anwenden, kommt es oft auch zu Angaben in verschiedenen Zahlensystemen. Hier eine Übersicht, was wem entspricht:

Zahlensystem	C	MODULA-2
dezimal	255	255
hexadezimal	0xFF	0FFH
oktal	0377	377B
LONG-Hex.-Werte	0xFFFFFFFFL	0FFFFFFFFH
LONG-Dez.-Werte	100000L	100000D

Das Casting, die Umwandlung einzelner Typen ist in beiden Sprachen ähnlich:

C	MODULA-2
char(Zeichen)	CHAR(Zeichen)
long(Langwort)	LONGINT(Langwort)

Zusätzlich gibt es im System-Modul des M2Amiga-Compilers die Funktion CAST, die Typen mit gleichem Speicherbedarf umwandelt.

Übrigens: Treffen Sie in einem C-Programm auf ein »Kaufmannsund« auch »Ampersand« (&) vor einer Variablen, so wird an dieser Stelle die Adresse der betreffenden Variable angegeben. In MODULA-2 erreichen Sie das durch die Funktion ADR.

Häufig werden auch Arrays bei der Programmierung benutzt. So sieht ein typischer C-Array aus:

```
char String[80];     (eindimensional)
char String2[3][80]; (zweidimensional)
```

In MODULA-2 würde das heißen:

```
String : ARRAY[0..79]         OF CHAR; (eindimensional)
String2: ARRAY[0..2],[0..79] OF CHAR; (zweidimensional)
```

Die Handhabung der Arrays ist in beiden Sprachen gleich:

C:

```
String[80]="z";
String2[1][80]="a";
```

```
MODULA-2:
```

```
String[79]:="z";
String2[0][79]:="a";
```

In einem C-Programm wird man oft auf seltsame Zeichen wie ¦¦, && oder ! treffen. Diese sind einfach logische Operatoren:

```
Operator     C       MODULA-2

UND          &&      AND oder &
ODER         ¦¦      OR
NICHT        !       NOT oder #
```

Dann unterscheiden sich noch die verschiedenen Kontrollstrukturen voneinander. Hier ist eine Gegenüberstellung:

Kontrollstruktur	C	MODULA-2

IF-Abfrage	if (a==b)
	{
	x=y;
	}
	else
	{
	y=x;
	}
		IF a=B THEN
		x:=y;
		ELSE
		y:=x;
		END;

WHILE-Schleife	while (a==b)
	{
	x=y;
		WHILE a=b DO
		x:=y;
		END;

Kontrollstruktur	'C	MODULA-2
	}	

REPEAT/UNTIL	do
	{
	x=y;
	}
	while (a<b);
		REPEAT
		x:=y;
		UNTIL a>b;

FOR-Schleife	for (x=1;x<10;x++)
	{
	y=x;
	}
		FOR x:=1 TO 9 DO
		y:=x;
		END;

LOOP/EXIT	for (;;)
	{
	...
	break;
	...
	}
		LOOP
		...
		EXIT;
		...
		END;

Fallunterscheidung	switch (x)
	{
	case 1 :Prog1;
	break;
	case 2 :Prog2;
	break;
	default:Prog3;
	break;
	}
		CASE x OF
		1: Prog 1|
		2: Prog 2|
		ELSE
		Prog3;
		END;

Die Routinen und Definitionen für die Systemprogrammierung unterscheiden sich namentlich und manchmal auch syntaktisch in einigen Fällen derart, daß eine komplette Gegenüberstellung hier unangebracht ist, doch sicher werden Sie auch mit diesem Anhang einiges anfangen können.

Nun zur Umsetzung von Basic-Programmen, zu der aber nicht allzuviel zu sagen ist, außer daß sie sehr, sehr schwierig ist, da Basic eine geradezu chaotische Sprache ist. Doch einige Dinge bezüglich der Typen seien hier aufgeführt:

```
&hffff entspricht 0FFFFH
&o7777 entspricht 7777B
```

```
x#     entspricht einem REAL-Wert x
a%     entspricht einem INTEGER- oder CARDINAL-Wert a
x&     entspricht einem LONGINT- oder LONGCARD-Wert x
x-     entspricht einem String (ARRAY OF CHAR) x
```

Versuchen Sie doch einfach mal, Programme anderer Sprachen (da empfiehlt sich natürlich C) aus z.B. Zeitschriften umzusetzen. Das trägt ungemein dazu bei, daß Sie mit der anderen Sprache und natürlich auch mit MODULA-2 vertraut werden. Beachten Sie dabei aber, daß das Umsetzen von Spracheigenheiten (z.B. Indirekte Adressierung mit Pointern in C, Inkrementieren von Pointern und Variablen in C) teils eine sehr genaue Kenntnis der Quellsprache erfordert, so sind teilweise Konstrukte der Sprache C nicht direkt nachzubilden.

Anhang C

Nützliche Public-Domain-Software

Sicher haben Sie schon mal von Public-Domain-Software (kurz PD) gehört. PD ist Software, die von den Autoren zum kopieren freigegeben wurde und die auf unzähligen Disketten gesammelt wurden. Es gibt viele verschiedene Serien, die sich aus 10–172 dieser Disketten zusammensetzen (und es werden immer mehr), zu den bekanntesten dürften FISH und RPD gehören. Disketten einer Serie können frei kopiert (bei Computerclubs z.B.) oder bei einem der zahlreichen PD-Händler in der Bundesrepublik bestellt werden. Da es verboten ist, Profit mit PD zu machen, darf der Preis nicht höher als die Materialkosten (Diskette, Porto u. Verpackung) sein. Sie sollten darauf achten, nicht mehr als 7 DM pro Diskette zu bezahlen und das auch nur bei absolut exzellenten Datenträgern, also 2DD, 135 TPI und 100% fehlerfrei. Da auf PD-Disks viele kleine und oft nützliche Programme, z.T. mit Quellcode vorhanden sind, bieten Sie für den Programmierer eine ausgezeichnete Wissensquelle. Einige zu den Themen in diesem Buch passende Disketten mit den nennenswertesten Programmen seien hier nun aufgeführt:

RPD 37	DUtil (Diskettenhilfsprogramm in Modula-2 mit TDI-Quellcode)
PANORAMA 12A	EGad (sehr guter Gadget-Editor, der C-Quellcodes erzeugt, die recht einfach umgesetzt werden können)
FISH 96	PopUpMenu (Pull-down-Menü an der aktuellen Mausposition, mit C-Quellcode)
	Chess (Schachprogramm ohne sonderliche Grafik und Mausbedienung, aber unglaublich spielstark, inklusive C-Quellcode)
FISH 30	MenuEditor (Menüeditor für C-Programme)
	FontEditor (Zeichensatzeditor)
	FineArt (viele sehr gute Bilder von Jim Sachs)
RPD 63	Fonts (mehr als 20 neue Fonts im Standard-Format)

FISH 113	M2Amiga (englische Demoversion des Compilers, zwar recht wenige Bibliotheken, aber ausreichend, um sich ein Bild des Systems zu machen)
RPD 112	NGI 1.3 (eins der besten PD-Programme, die ich kenne: IFF-Brushes von DPaint II können in Icons, Bobs, VSprites, Images und vieles mehr umgewandelt werden. Das Programm erzeugt zwar C-Quellcode, aber das ist nicht weiter tragisch, da es sich mit dem Aufwand, der für die Umsetzung nach Modula-2 nötig ist, in Grenzen hält.)
Amok 1–14	(Auch in der KICKSTART-Serie): Vierzehn Disketten, die randvoll mit guten Modula-Quellcodes sind. Unbedingt ansehen!

Anhang D

Der Benchmark- und TDI-Compiler

Neben dem MODULA-2-Compiler M2Amiga existieren noch der Benchmark- und TDI-Compiler, die sich in manchen Dingen unterscheiden. In diesem Buch wird hauptsächlich auf den M2Amiga-Compiler der Firma A+L Meier-Vogt eingegangen. Es ist ein wenig Arbeit erforderlich, um die Beispielprogramme umzuschreiben, denn die Module sind recht unterschiedlich implementiert:

- Die Feldnamen und Konstanten fangen beim Benchmark- und TDI-Compiler ausnahmslos mit Großbuchstaben an und unterscheiden sich manchmal in der Definition. Näheres können Sie nur durch Studium Ihres Compiler-Handbuches herausfinden

- Die Bibliotheken müssen vor Gebrauch geöffnet werden.

- Die Definitionsmodule unterscheiden sich ungemein. Während es beim M2Amiga-Compiler verhältnismäßig wenige, dafür aber sehr umfangreiche Definitionsmodule gibt, existieren beim Benchmark- und TDI-Compiler sehr viele, weniger umfangreiche. Wo Sie nun eine gewünschte Konstante, Funktion oder Struktur finden können, steht im Handbuch Ihres Compilers.

- Der IntuitionBase-Record ist bei TDI nicht vollständig implementiert.

- Die Operatoren REM und / funktionieren bei TDI bei CARDINALs und LONGCARDs nicht.

- Das Modul Coroutines fehlt beim Benchmark-Compiler.

- Das Modul FileSystem fehlt beim TDI-Compiler.

Anhang E

Literaturverweise

In diesem Anhang möchte ich Sie noch auf einige empfehlenswerte Bücher zum AMIGA hinweisen. Ich habe Wert darauf gelegt, nicht zehn Seiten mit Büchern, die sowieso keiner braucht vollzuschreiben, nur weil sie irgendetwas mit den hier behandelten Themen zu tun haben, sondern wirklich nützliche Bücher aufgeführt. Sie brauchen wirklich nicht alle Werke zu haben, zumal Computer-Bücher meist recht teuer sind, doch bieten sie ganz nützliche Zusatzinformationen.

Commodore Business Machines, Inc.
ROM KERNEL REFERENCE MANUAL: LIBRARIES AND DEVICES
Addison-Wesley Publishing Company, Inc.
ISBN 0-201-11078-4

Ausführliche englische Beschreibung der Library-Funktionen und Devices von Kickstart V1.1. Das Werk ist leider veraltert und nicht immer leicht verständlich, doch bietet es eine Fülle von Informationen für C-Programmierer. Können Sie C, so wird es Ihnen nicht allzu schwer fallen, die Beispielprogramme umzusetzen. Vieles, was in diesem Buch steht (und auch vieles, was nicht darin steht) wurde aber schon hier besprochen, so daß jeder für sich entscheiden muß, ob sich die Anschaffung dieses Werkes lohnt (Standarddokumentation des AMIGA Betriebssystems).

Commodore Business Machines, Inc.
ROM KERNEL REFERENCE MANUAL: EXEC
Addison-Wesley Publishing Company, Inc.
ISBN 0-201-11099-7

Ausführliche englische Beschreibung des EXEC-Systems unter Version v1.1 sowie der verschiedenen Formen des IFF-Standards. Es gilt dasselbe wie für das vorherige Buch, was die Sprache und den Inhalt betrifft (Standarddokumentation des Amiga Betriebssystems).

Commodore Business Machines, Inc.
INTUITION REFERENCE MANUAL
Addision-Wesley Publishing Company, Inc.
ISBN 0-201-11076-8

Ausführliche englische Beschreibung von Intuition, doch leider ohne Beispielprogramme. Was Sprache und Inhalt betrifft, siehe »ROM KERNEL REFERENCE MANUAL: LIBRARIES AND DEVICES«
(Standarddokumentation des Amiga Betriebssystems).

Commodore Business Machines, Inc.
HARDWARE REFERENCE MANUAL
Addison-Wesley Publishing Company, Inc.
ISBN 0-201-11077-6

Verhältnismäßig knappe und lückenhafte, englische Beschreibung der Hardware des Amiga. Dieses Werk ist wohl nur für passionierte Assembler-Programmierer interessant, doch für die gibt es bessere Werke, siehe »AMIGA INTERN« (Standarddokumentation des Amiga Betriebssystems).
(Alle vier Bände des ROM-KERNEL-MANUALS sind zusammen zum Preis von ca. 240 DM erhältlich.)

Dittrich/Gelfand/Schemmel
AMIGA INTERN
Data Becker
ISBN 3-89011-104-1
69,- DM

Ausführliche Besprechung der Amiga-Hardware (Programmierung in Assembler), des EXEC-Systems (Programmierung in C) sowie einiger anderer Dinge (IFF, DOS, ...). Dieses Werk ist nicht unbedingt notwendig, da eigentlich alles, was für den MODULA-2-Programmierer interessant ist, schon hier erwähnt wurde.

Weltner/Trapp/Jennrich
SUPERGRAFIK
Data Becker
ISBN 3-89011-254-4
59,- DM

Ausführliche Besprechung der Programmierung von GRAPHICS in Basic und C. Nicht unbedingt notwendig.

Krüger
Amiga Programmieren mit MODULA-2
Markt & Technik
ISBN 3-89090-554-4
69,- DM

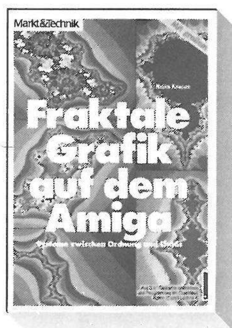
Behutsame Einführung in die Sprache MODULA-2 eine kurze Einführung in die Programmierung mit Intuition. Als Einsteiger-Buch sehr empfehlenswert, inklusive Diskette mit allen Beispielprogrammen.

Stichwortverzeichnis

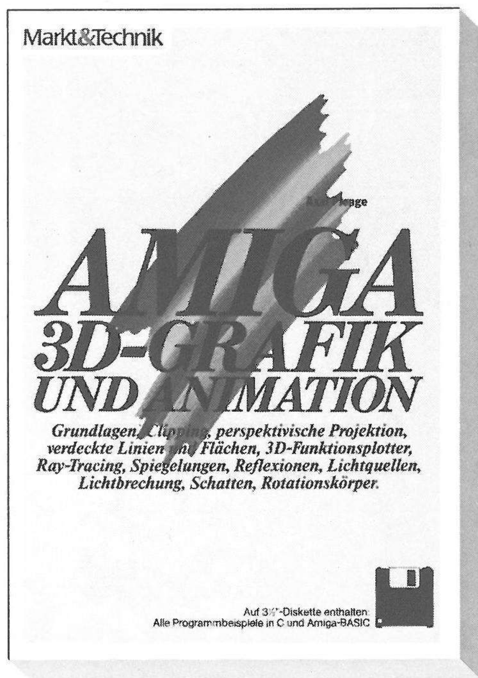
S

T

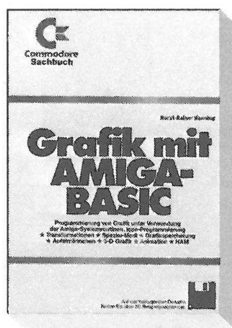
Bücher zum Amiga



H. Knappe
Fraktale Grafik auf dem Amiga
Das Thema dieses Buches wird den meisten als ungewöhnlich erscheinen, denn es führt in die Grenzbereiche des heutigen Wissens der Mathematik und Technik.
Grundlegende Kenntnisse der Programmiersprache C und ihrer Anwendung auf dem Amiga werden vorausgesetzt. Für die nachträgliche Veränderung berechneter Bilder ist es sinnvoll, ein Malprogramm (z. B. Deluxe Paint) zu besitzen.
1988, 278 Seiten, inkl. Diskette
Bestell-Nr. 90600
ISBN 3-89090-600-1
DM 79,–
(sFr 72,70/öS 616,-)



A. Plenge
Amiga 3-D-Grafik und Animation
Angefangen bei den einfachsten Problemstellungen lernen Sie, professionelle 3-D-Grafiken auf Ihrem Commodore Amiga zu planen, zu programmieren und darzustellen.
1988, 376 Seiten, inkl. Diskette
Bestell-Nr. 90526
ISBN 3-89090-526-9
DM 69,–
(sFr 63,50/öS 538,-)



H. R. Henning
Grafik mit Amiga-Basic
Dieses Buch ist speziell der Grafik-Programmierung auf dem Amiga gewidmet. Der erste Teil stellt für den Anfänger alle bekannten Grafik-Befehle des Amiga-Basic vor. Mit Beginn des zweiten Teiles werden die Routinen des Betriebssystems zur Grafik-Programmierung herangezogen. Damit werden die Möglichkeiten des Basic um ein Vielfaches erweitert, und es sind Geschwindigkeiten möglich, die kaum vermuten lassen, daß dabei ein Basic-Programm abläuft.
1989, ca. 300 Seiten, inkl. Diskette
Bestell-Nr. 90669
ISBN 3-89090-669-9
DM 59,–
(sFr 54,30/öS 460,-)

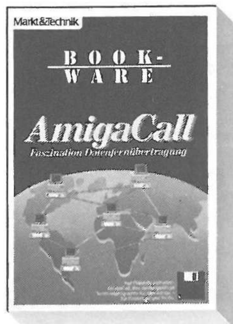


Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

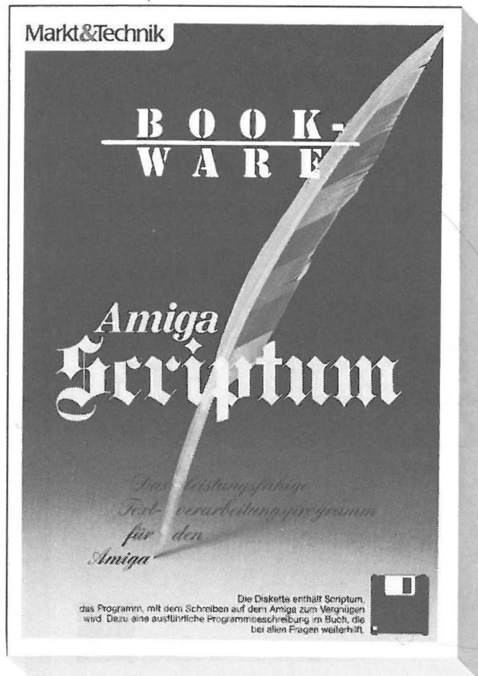
429/901

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

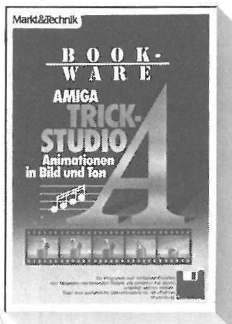
Bücher zum Amiga



Atlantis
AmigaCall
Treten Sie ein in die faszinierende Welt der Datenfernübertragung. Kommunizieren Sie über Mailboxen mit erfahrenen Computer-Anwendern, die Ihnen bei Ihren Problemen weiterhelfen können, oder Sie erhalten auf diesem Wege leistungsfähige Public-Domain-Software. AmigaCall nimmt Ihnen die meiste Arbeit ab. Schließen Sie Ihr Modem oder Ihren Akustikkoppler an, starten Sie AmigaCall - und auf geht's.
1988, 133 Seiten, inkl. 3¹/₂"-Programmdiskette
Bestell-Nr. 90716
ISBN 3-89090-716-4
DM 99,-*
(sFr 91,-*/öS 842,-*)



R. Arbinger/I. Krüger
Scriptum
Scriptum - das schnelle, leistungsfähige Textverarbeitungssystem für den Amiga. Ausführliche Bedienungsanleitung im Buch. Für alle, die auf dem Amiga Texte verarbeiten wollen.
1989, ca. 200 Seiten, inkl. 3¹/₂"-Programmdiskette
Bestell-Nr. 90650
ISBN 3-89090-650-8
DM 79,-*
(sFr 72,70*/öS 672,-*)
* Unverbindliche Preisempfehlung



Atlantis
Trickstudio A
Ob Sie Computerfilm-Pionier sind oder Trickprofi, ob Sie von Walt Disney inspiriert sind oder einfach nur einen guten Lehrfilm für technische Abläufe benötigen: Mit Trickstudio A können Sie Ihre eigenen Trickfilme erstellen und diese mit Sound und Geräuschen untermalen.
Wie wäre es also mit einem Stummfilm-Slapstick, einem Krimi oder einem Werbefilm für Ihr Schaufenster? Dazu Ihre Lieblingsmusik oder digitalisierte Stimmen? Entwerfen Sie die Einzelbilder, z. B. mit Deluxe Paint, erstellen Sie eine Sounddatei und dann: Klappe - Film; die erste.
1988, 86 Seiten, inkl. 3¹/₂"-Programmdiskette
Bestell-Nr. 90715
ISBN 3-89090-715-6
DM 99,-*
(sFr 91,-*/öS 842,-*)



Zeitschriften · Bücher
Software · Schulung

427/901

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bitte schneiden Sie diesen Coupon aus, und schicken Sie ihn in einem Kuvert an:
Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar

Computerliteratur und Software vom Spezialisten

Vom Einsteigerbuch für den Heim- oder Personalcomputer-Neuling über professionelle Programmierhandbücher bis hin zum Elektronikbuch bieten wir Ihnen interessante und topaktuelle Titel für

• Apple-Computer • Atari-Computer • Commodore 64/128/16/116/Plus 4 • Schneider-Computer • IBM-PC, XT und Kompatible
sowie zu den Fachbereichen Programmiersprachen • Betriebssysteme (CP/M, MS-DOS, Unix, Z80) • Textverarbeitung • Datenbanksysteme • Tabellenkalkulation • Integrierte Software • Mikroprozessoren • Schulungen.
Außerdem finden Sie professionelle Spitzen-Programme in unserem preiswerten Software-Angebot für Amiga, Atari ST, Commodore 128, 128 D, 64, 16, für Schneider-Computer und für IBM-PCs und Kompatible!
Fordern Sie mit dem nebenstehenden Coupon unser neuestes Gesamtverzeichnis und unsere Programmservice-Übersichten an, mit hilfreichen Utilities, professionellen Anwendungen oder packenden Computerspielen!



709005

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2,
8013 Haar bei München, Telefon (089) 4613-0

Adresse:

Name

Straße

Ort

Bitte schicken Sie mir:
☐ Ihr neuestes Gesamtverzeichnis
☐ Eine Übersicht Ihres Programmservice-Angebotes aus der Zeitschrift
☐ Außerdem interessiere ich mich für folgende/n Computer:

(PS: Wir speichern Ihre Daten und verpflichten uns zur Einhaltung des Bundesdatenschutzgesetzes)

Markt&Technik Verlag AG
– Unternehmensbereich Buchverlag –
Hans-Pinsel-Straße 2
D-8013 Haar bei München